2017 IEEE Symposium on Security and Privacy

# An Experimental Security Analysis of an Industrial Robot Controller

Davide Quarta*, Marcello Pogliani*, Mario Polino*,
Federico Maggi*†, Andrea Maria Zanchettin*, and Stefano Zanero*
*Dipartimento di Elettronica, Informazione e Bioingegneria – Politecnico di Milano, Italy
{*davide.quarta, marcello.pogliani, mario.polino, andreamaria.zanchettin, stefano.zanero*}*@polimi.it*
†Trend Micro Inc.
*federico_maggi@trendmicro.com*

***Abstract*—Industrial robots, automated manufacturing, and efficient logistics processes are at the heart of the upcoming fourth industrial revolution. While there are seminal studies on the vulnerabilities of cyber-physical systems in the industry, as of today there has been no systematic analysis of the security of industrial robot controllers.**

**We examine the standard architecture of an industrial robot and analyze a concrete deployment from a systems security standpoint. Then, we propose an attacker model and confront it with the minimal set of requirements that industrial robots should honor: precision in sensing the environment, correctness in execution of control logic, and safety for human operators.**

**Following an experimental and practical approach, we then show how our modeled attacker can subvert such requirements through the exploitation of software vulnerabilities, leading to severe consequences that are unique to the robotics domain.**

**We conclude by discussing safety standards and security challenges in industrial robotics.**

## I. Introduction

Industrial robots are mechanical, multi-axis "arms" used mainly in the manufacturing sector, or for automation in general. The International Federation of Robotics forecasts that, by 2018, approximately 1.3 million industrial robot units will be employed in factories globally, and the international market value for "robotized" systems is approximately of 32 billion USD [1]. In all of their forms, robots are complex automation devices that heavily interact with the physical world—in this sense they are cyber-physical systems (CPSs)—and include multiple hardware and software components: mechanical actuators, controllers, sensors, human-interaction devices, control logic, firmware, and operating systems. Moreover, the growing integration of computerized monitoring of physical production processes leads to robots being interconnected among themselves and with external services. For instance, in the "Industry 4.0" vision, an enterprise-management system automatically orders any part needed to complete the scheduled production, reconfigures the robotized production lines, and tracks their operational status [2].

Along with the major improvements to safety, efficiency, and production time, this increased complexity and interconnection offers a novel attack surface, with consequences ranging from the compromise of the controlling machines up to effects on the production chain. One may even conceive that, in the future, a manufacturer could leverage these novel attack opportunities to affect the reputation of a competitor—not to mention the possibility that enemy nations could attack each others' factories manufacturing critical goods [3].

A further exacerbating factor is that robot controllers cannot be promptly patched, since updates may require unacceptable downtime, or even introduce regressions and new software bugs that render the software unusable. This "patching problem" makes the exploitation window of a vulnerability much longer, eventually increasing the impact of an attack.

Taking advantage of new interconnections to compromise devices originally designed to work in isolation is a pattern already observed, for instance, in the automotive [4], [5] and industrial control system (ICS) sectors. After Stuxnet [6], other successful attacks have been recently observed: In 2014, an attack on a German steel mill caused the inability to shut down a blast furnace. In 2015, 295 security incidents were reported to the U.S. ICS CERT [7], of which 22 reached the core of critical control systems.

Unfortunately, even a simple Shodan query (Section II) shows that sometimes industrial robots are exposed on the Internet without being properly secured. Alarmed by this finding, we ran a survey among robot users, and discovered that they either do not regard a security incident involving a robot as realistic, or are not fully aware of the potential damages.

Various informal communications [8] already discussed the risks caused by security vulnerabilities in industrial robots, highlighting lack of awareness by both developers and public. More recently, researchers have raised concerns about the cyber attack resiliency of unmanned aerial vehicles (UAVs) [9] and robots for tele-operated surgery [10]. However, the main focus was the need for secure communication protocols [11], without analyzing system-specific attacks.

To the best of our knowledge, there is no systematic analysis of the attack surface and of the *impact* of cyber attacks against industrial robots enabled by software vulnerabilities and architectural flaws. In this paper, we systematically analyze the feasibility of attacking a modern industrial robot by exploring concrete attack vectors that, when exploited, can subvert the interaction between a robot and the surrounding


DOI 10.1109/SP.2017.20

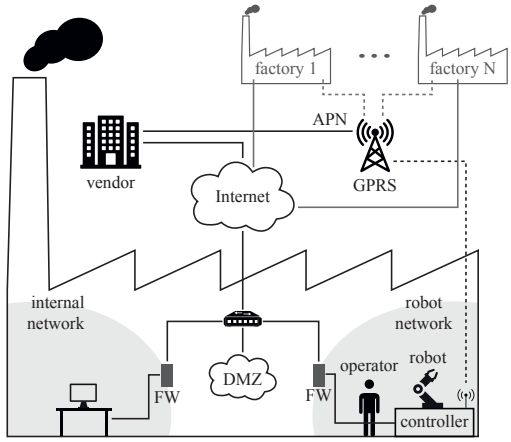



Figure 1. A high-level depiction of an Industry 4.0 ecosystem, to help the reader contextualize our work.

environment, thereby violating its basic requirements. In other words, we wonder to what extent, starting from the exploitation of the "cyber" components of a robot, an attacker can affect the physical environment. To answer this question, we propose a domain-specific attacker model, discuss how certain combinations of software vulnerabilities enable classes of attacks unique to industrial robots (e.g., circumventing safety measures, impairing the precision of movements), and evaluate their potential impact. More specifically, we enumerate five classes of attacks, based on the observation that a robot working under normal circumstances should at least be able to read accurately from sensors, execute its control logic, perform precise movements, and not harm humans.

To show the feasibility of the attacks, we present a case study on a modern industrial robot sold by a major manufacturer. Due to standards and architectural commonalities among most modern industrial robots, the robot we chose is representative of a large class of industrial robots. Guided by our attacker model, we discover various vulnerabilities that allow an attacker to completely and remotely compromise the robot-controlling computers, and show how we used them to implement some of the proposed classes of attacks.

In summary, we present the following contributions:

- We define an attacker model for industrial robots, describing the capabilities of an adversary to successfully develop and convey attacks;
- Starting from the "core" architectural features of a generic industrial robot, we describe concrete, robot-specific attack classes;
- We conduct an experimental security assessment on a defacto-standard robot;
- We analyze how, concretely, an attack can leverage software vulnerabilities to carry out robot-specific attacks, evaluating their impact, and discussing the future security challenges.

## II. Context and Motivation

We contextualize our work using a high-level Industry 4.0 example scenario (Figure 1), which derives from our domain knowledge, the standard architectures of industrial robots (Section IV), and some of the answers to our survey (summarized in Section II). We intentionally designed this scenario to be vendor-agnostic and simple, while representing a complete and realistic deployment: Robots are connected to their network via a controller, which can be approximated as a computer that controls the robot and includes human-interaction subsystems (e.g., joysticks, switches, I/O and diagnostic ports). In the best scenario, robots are connected to an isolated subnet. However, we found out that such subnet is still often remotely accessible, either through a connection to the Internet, or via dedicated vendor access (e.g., via GPRS).

The motivation for this work stems from the following key observations:

**Interconnected Robots.** Industrial robots are "connected" primarily for programming and maintenance purposes—a use case specified by ISO standards [12]. For instance, in a large car production plant developed by KUKA Robotics, all the 259 robots are connected to central control and monitoring systems [13]. The industrial robot ecosystem is also emphasizing a richer human-robot interaction (HRI) and complex APIs to integrate robots and ICT ecosystems. For example, ABB controllers expose a Robot Web Service API [14], an HTTP REST API that allows external programs to speak with the robot controller. The availability of easy-to-use APIs has led to the creation of intermediate layers that allow the control of robots, even from consumer-grade devices such as smartphones [15], [16].

It is not hard to envision a pervasive future for industrial robots, because they are following the same natural evolution that characterized other digital devices such as smartphones. In 2011, the Robot App Store (http://www.robotappstore.com/), an application-distribution platform supporting many commercially available consumer robots and drones, opened to the public. ABB hosts the RobotApps forum (https://robotapps.robotstudio.com), where users can exchange 3D models, videos, add-ins and "apps".

***First Observation.*** The increased connectivity of computer and robot systems is (and will be) exposing robots to cyber attacks. Indeed, nowadays, industrial robots—originally conceived to be isolated—are exposed to corporate networks and to the Internet.

**Software-defined Safety.** The design of industrial robots emphasizes safety concerns: Robots are traditionally designed to operate in a workspace physically separated from humans (e.g., a cage). However, vendors are introducing several models of *collaborative robots* (or cobots), able to work nearby humans (e.g., ABB's YuMi, FANUC's CR-35iA [17], and Universal Robots' cobots). This, alongside with the gradual shift of safety devices' implementations from hardwired logic to more flexible software-based implementations, increases the relevance of safety concerns.

***Second Observation.*** Safety is of increasing importance in robotics, but the implementation of safety mechanisms in software is increasing the potential impact of security issues:

An attacker may disrupt operations and indirectly pose safety threats to human operators.

**Robot Users and Cybersecurity.** To understand how much industrial robotics practitioners and researchers are aware of cyber security risks deriving from industrial robots, we performed a preliminary investigation.

We sent a survey to 50 domain experts, both from academia (i.e., members of the automatic control and robotics community) and from the industry, including representatives of relevant scientific and technical societies. We received 20 answers, and we manually followed up to investigate some of them further. Appendix A details the results of this survey. The proposed questions (common for all subjects) were aimed at understanding how industrial robots are deployed, how safety measures are used, how robots are programmed, and, in general, to obtain a picture of the users' perception of security risks. About 30% of the participants found the default safety measures of the deployed robots "too limiting" for specific use cases, and, in fact, 60% told us that they had to customize them. The answers also unveiled issues related to the development cycle of robot programs: In many cases, employees are not accountable for changes to the robot program code. Regarding security practices, 28% of the respondents did not enforce access control policies on the robot's controller, 30% had robots accessible from the Internet, and 76% never performed a professional cyber security assessment on their infrastructure. More than 50% of the respondents did not consider cyber attacks a realistic threat, and those who do, were mainly concerned about safety issues.

***Third Observation.*** According to our survey, awareness of security risks seems scarce. Moreover, as detailed in Section III, we found evidence of 28 robots directly exposed to the Internet, with an accessible FTP server.

Given the premises outlined in this section, we believe that an experimental security analysis of industrial robots is beneficial for both the research community and the industry.

Therefore, in this work, we (a) define an attacker model; (b) introduce industrial-robot-specific attacks based on the tasks that a robot must be able to perform, and, in general, on the properties that it must possess; and (c) experimentally verify the feasibility of such attacks on a standard robot architecture under the defined attacker model.

## III. Attacker Model

We model attackers according to their *goals*, their level of *access* to the system, and their *capabilities*.

### A. Attacker Goals and Threat Scenarios

The attacker wants to obtain control of the target industrial robot as a means to carry out attacks, leveraging the unique fact that the system is interacting with the physical world. We reason about attacker profiles through four example threat scenarios (which are not necessarily the only possible ones).

***Production Outcome Altering.*** The attacker may want to inject faults and micro-defects in the production. These defects can cause immediate or delayed financial loss, and damaged reputation, resulting in an advantage for competitors and potentially damaging the brand reputation. Depending on the manufactured goods, defects can also cause fatalities (e.g., in automotive, transportation or military fields).

***Physical Damage.*** An attacker could damage machinery, or cause injuries to people working in the factory, for instance by disabling or substantially altering safety devices.

***Production Plant Halting.*** According to the extent of the damages caused by the cyber-attack, the production may or may not be promptly restarted (e.g., the time to repair varies greatly). Although financial losses due to downtime are difficult to estimate, and vary greatly according to the type and size of the targeted company, the vice president of product development at FANUC stated [18] that "unplanned downtime can cost as much as $20,000 potential profit loss per minute, and $2 million for a single incident."

***Unauthorized Access.*** An attacker may want to steal sensitive data (e.g., source code; information about production schedules and volumes).

### B. Access Level

We broadly distinguish between network and physical attack vectors. We focus exclusively on accessible hardware components that allow access to the digital attack surface, ignoring attacks that involve breaking or tampering with the physical security of the robot controller's case.

**Network Attacker.** Even when the robot is not directly exposed to the Internet, industrial robot controllers can be connected to the factory LAN, or to (vulnerable) remote service facilities. The techniques and tactics used to gain network access are beyond the scope of this work. However, an attacker usually leverages various entry points to compromise a computer connected to the factory network, even resorting to offline methods (e.g., USB sticks [19]) used to pivot attacks against internal devices.

***Remote Exposure.*** We found that, sometimes, controllers are reachable from the outside. For about two weeks, we monitored Shodan and ZoomEye, two search services that index data from Internet scans, repeatedly querying them for string patterns contained in the FTP banner of the top robot manufacturers (e.g., "ABB Robotics," "FANUC FTP," as detailed in Table I). Despite we limited our search to the top manufactures and to FTP, the findings support our remote-attacker assumption. We found three distinct Internet-exposed ABB controllers, one of them providing unrestricted access using anonymous credentials (i.e., the authentication system was disabled). For ethical reasons, we did not directly attempt to connect to those systems; instead, we used only the data provided by the search services to filter out false positives.

Some industrial robots embed proprietary remote access devices, used, for example, by the vendor for remote monitoring and maintenance. Such devices are *industrial routers*, often dubbed with vendor-specific terms such as "service box." The connection between the industrial router and the remote

Table I
INTERNET-EXPOSED INDUSTRIAL AUTOMATION DEVICES, ACCORDING TO ZOOMEYE AND SHODAN SEARCH RESULTS

| Search string | Entries | Country |
|---|---|---|
| ABB Robotics | 5 | DK, SE |
| FANUC FTP | 9 | US, KR, FR, TW |
| Yaskawa | 9 | CA, JP |
| Kawasaki E Controller | 4 | DE |
| Mitsubishi FTP | 1 | ID |

service center can happen over the Internet, through a VPN, or through a GPRS network that can use the commodity carrier-provided APNs, or vendor-specific APNs configured on M2M SIMs. In the latter case, if not properly configured, all factories using robots from the same vendor will share the same APN and will be able to connect to one another.

Industrial routers provide a helpful attack surface to gain access to a robot controller. For example, an attacker could target a widespread vendor of such appliances, whose products are also resold by robotics OEMs as part of their support contracts. Among these vendors, eWON is quite representative. A simple Shodan query for the *default* banner of the embedded web server (`Server: eWON`) yielded 1,044 results, without accounting for customized banners. The web-based configuration console is easily "fingerprintable," and attackers could exploit vulnerabilities or misconfigurations in the router to gain access to the robot. For example, we analyzed a eWON device in a black-box fashion and discovered a severe authentication-bypass vulnerability that allows an attacker to read the configuration and device information (e.g., event logs).

**Physical Attacker.** The simplest and most common type of physical attacker is the *robot operator*, who uses the robot's handeld HRI interface (i.e., a joystick with a touchscreen display) on a regular basis to program the robot or manually pilot its arm(s).

A slightly more sophisticated profile is the *"casual" attacker* (e.g., malicious contractor or technician), who is able to plug a device into the robot controller's openly accessible RJ-45 (or equivalent) port. This grants full access to the robot controller's computer via Ethernet or other I/O interfaces. As detailed in Section IV, even standard Ethernet access can mean that the attacker is allowed to send and receive network frames to and from various sub-systems of the controller through an unfiltered connection. The physical attacker can leverage further vectors, for instance internal I/O interfaces that the controller uses to communicate directly with the robot's components (e.g., DeviceNet over CANbus, as found on ABB's controllers). Thus, a casual physical attacker is strictly more powerful than a network attacker.

### C. Attacker Characterization

**Attacker Profile.** Following and simplifying the taxonomy introduced in [20], the most likely profile for attacks requiring physical access is an *insider*, whereas network attacks can be performed by a broader set of *cybercriminals*. Given the safety-critical and economic characteristics of the targets, we need also to consider nation-state level attackers. To model them, we assume that they would be able to reach the same access level and target knowledge typical of an insider, while, at the same time, would be able to launch sophisticated attacks from remote endpoints. Such attackers would also not be constrained by costs.

**Technical Capabilities.** We assume attackers to be familiar with the structure of the target industrial robot, and to possess the technical skills to perform reverse engineering without exploiting any insider technical knowledge. Realistically, they can rely on publicly available information (e.g., controller software and firmware available for download from the vendor's website), and some reverse engineering. As a matter of fact, we learned most of the details described in this paper by reading freely available technical documentation. Therefore, an attacker can do the same.

**Access to Equipment.** We consider it to be trivial for an attacker to access a copy of the controller's firmware binary executables in order for them to reverse engineer the software and discover vulnerabilities. Indeed, some vendors make the controller firmware, or simulation environments, freely available for download from their website (e.g., the controller's firmware for ABB robots is included in the RobotWare distribution as part of the RobotStudio suite).

Depending on the attackers' budget, they may or may not be able to test exploits before carrying out an attack, as this would require access to a full-fledged deployment. If unable to access a real robot, an attacker can leverage simulators distributed by vendors in order to gain technical knowledge about the target, and to prepare an attack payload. For example, ABB's RobotStudio suite allows one to make use of a simulated environment. The simulator is contained in a shared library that shares most of the code (and, thus, most of the vulnerabilities) with the firmware of the controller's computer. This is clearly an advantage to the attacker, who can gather complete access to platform-specific details without accessing an actual controller.

On the other hand, some vendors (e.g., COMAU and Kuka) provide their software only to customers. In this specific case, the attacker needs at least temporary physical access to a robot controller to dump a copy of the firmware. When not uploaded remotely, the firmware is usually loaded from a removable medium such as a MMC, which can be leaked—or stolen—from the company that has access to the robot's software as part of a support contract. This does not necessarily require an insider attacker, but just minimal knowledge about the controller, and short-term network or physical access to the robot.

Similarly to other specialized CPS domains (e.g., avionics and ICS), used or reconditioned industrial robot parts are available for sale without restrictions. The cost of such parts may vary, and a complete robot with its controller has a relatively high price tag. However, we do not consider such a price out of reach for our attacker. For example, search-

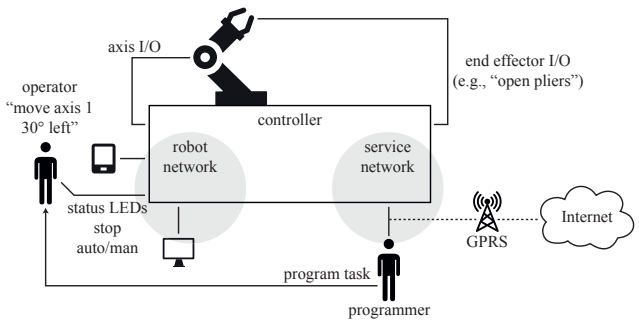



Figure 2. Black-box view of a standard industrial robot architecture. The programmer or the operator issue high-level commands to the controller (e.g., via a REST API, with a program on the HRI interface, moving the joystick). The controller translates such commands into low-level inputs for the actuators (e.g., end effectors, servo motors) through dedicated I/O interfaces. The controller is also reachable through a remote-access interface.

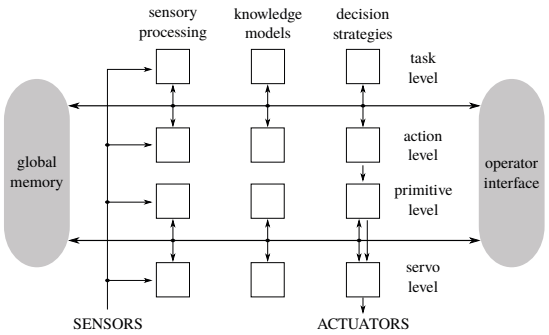



Figure 3. Abstract representation of the control system architecture of a robot controller. The arrows represent the data flow, and the blocks represent "generic" modules. From left to right, the controller acquires measurements from the environment, uses its knowledge and internal state to process the inputs, takes a decision, and actuates it. The functionality of the robot can be expressed at various levels of abstraction, from the highest level at the top (task, e.g., what the operator wants the robot to do), to the lowest level at the bottom (servo, e.g., current to apply to the motor's driver).

ing online marketplaces (e.g., globalrobots.com, ebay.com, alibaba.com) shows that the IRB 140 manipulator from ABB, which matches the reference setup we used in our experimental analysis, can be purchased for a price ranging from $13,000 to $28,950 together with an outdated S4C controller; other manipulators with an IRC5 controller can be found in the price range of $24,999 to $35,500. Ultimately, access to specific features (e.g., the GPRS remote service box of the robot that we analyzed) is more complex, as these are only available directly from the vendor as part of support contracts.

## IV. Industrial Robots Architectures

This section describes the basic structure of an industrial robot, including common components, the tasks it can perform, and how such tasks translate from high-level commands to concrete actions. A visual overview of a generic industrial robot system is given in Figure 2.

**Scope.** The scope of our analysis is defined by all the components that *must* be part of a standard industrial robot. For instance, we exclude the overall control of the manufacturing process, as it involves multiple external control systems and parts that would require a per-deployment analysis in order to obtain a thorough security assessment.

The only optional components in our scope are remote-access devices, which are commonly offered by vendors. This decision is due to the security relevance of such devices, and to account for remote adversaries in the attacker model.

### A. Architecture, Components and Functionality

Industrial robots are extensively standardized [21]: They are architecturally, functionally, and technically similar across vendors, and share a minimum set of requirements.

**Robot.** An *industrial robot* is an "automatically controlled, re-programmable, multipurpose manipulator programmable in three or more axes, which can be either fixed in place or mobile for use in industrial automation applications" (ISO 8373 [22]). Mechanically, an industrial robot is an arm with two or more joints, terminated by an *end effector* (e.g., pliers, cutter, laser beam welder) that interacts with the environment. The main characteristic of an *industrial* robot is its multipurpose nature, which implies a high level of complexity in the controller.

**Controller.** The robot *controller* is a complex device, typically enclosed in one or more chassis, "hiding" a multitude of interconnected electrical and computer systems. A robot controller is designed with an emphasis on efficiency, complex motion description, nonlinear control, and interaction with human operators.

By exploring the current product lines from leading vendors such as ABB, COMAU, Yaskawa, FANUC, and Kuka, we confirm that they all share a common architecture, types of components, and functional characteristics. We noticed that most of the controllers on the market also implement a similar software architecture, with operator interfaces running on Windows CE-based embedded systems (or equivalent), and one or more real-time VxWorks-based controller computers.

**Control System.** The controller implements a *control system* to supervise the robot's activities, making it one of the most safety- and security-critical components: It embodies the logic and functionality required to monitor and pilot the mechanical parts of the robot, and to communicate with the environment.

As a reference, throughout this paper, we will refer to the functional model proposed in [23] and depicted in Figure 3. Horizontally, the control system is split into three functional modules, and four hierarchical levels. The *sensory processing* modules capture the state of the system (e.g., via various sensors and servo motors position). The *knowledge models* modules hold the "knowledge" of the system and of the environment. The *decision strategies* modules translate high-level tasks into actions. Vertically, the *task level* consists of the task specifications given by the user. The *action level* translates the symbolic commands from the task level into motion paths. The *primitive level* computes the motion trajectory and manages the control strategy. Finally, the *servo level* reads measurements from sensors (e.g., position, velocity, torque),

and engages the control algorithms to provide a drive signal to the servo motors.

**Human-Robot Interaction.** The human *operator* is in charge of monitoring, starting, and stopping the operations of the robot, whereas the *programmer* is in charge of writing *task programs* (i.e., the set of instructions that define the specific task of the robot).

The interaction between human and robot via a user interface is generically called *human-robot interaction* (or HRI). For example, operators and programmers can monitor the status of the robot, and program it through the *teach pendant*, a hand-held unit connected to the controller via wired or wireless connection (Figure 4b shows a real pendant by ABB). The teach pendant looks like a "heavy duty" tablet, augmented with physical buttons and joysticks. In Figure 3 the HRI interfaces (the teach pendant or the operator's computer running specific control software) are generically represented by the "operator interface."

The operator needs to interact with multiple functional levels in order to control the robot (e.g., to program tasks, to modify configurations, and to stop servomotors in case of an emergency), and to be informed on the system state for supervision and intervention (e.g., which motor is moving, is the robot in manual or automatic mode?). Moreover, the operator can permanently modify the robot's knowledge (e.g., load a program, move the robot's arm), which in turn changes the decisions taken by the control system.

**Robot Programming.** The controller accepts task specifications written using a domain-specific programming environment. These programs can be written online or offline.

*Online* programming is common in modern robotic applications: The robot is "piloted" to record a sequence, by moving (specifically, *jogging*) the axes in the desired position using the teach-pendant's joystick, and storing the recorded data (e.g., coordinates of the position) read via the joint position transducers. This technique is called "teaching by showing."

*Offline* programming environments allow one to build more complex programs on a simulated robot replica. Although source code can be written with text editors (see Appendix C), programming is aided by simulators (e.g., ABB's RobotStudio or Universal Robot's URSim), which offer a software representation of the robot, allowing programmers to experiment with new features, and write and debug programs in a safe environment. The resulting program can be loaded on the real controller.

Robot programs are stored in the controller's *global memory*. It holds the functional blocks needed to exchange information between levels and modules, and maintain estimations of the state of the whole system (e.g., known position on each axis) and of the environment (e.g., temperature, weight of the piece held by the pliers).

**Automatic vs. Manual Mode.** The controller can work in *automatic* or *manual* mode. In automatic mode—intended for regular operations of the robot in production—, the controller loads and executes task programs from the *global memory*; in manual mode, the robot performs movements according to inputs issued by the operator through the teach pendant. Manual mode allows both a reduced-speed mode, used for programming the robot, and a high-speed mode, used for testing.

Table II
ATTACK SUMMARY WITH VIOLATED REQUIREMENTS

| Attack | Safety | Integrity | Accuracy |
|---|---|---|---|
| Control Loop Alteration | ✓ | ✓ | ✓ |
| *User-perceived* Robot State Alteration | ✓ | ✗ | ✗ |
| Robot State Alteration | ✓ | ✓ | ✓ |
| Production Logic Tampering | ✓ | ✓ | ✓ |
| Calibration Parameters Tampering | ✓ | ✓ | ✓ |

### B. Summary of Requirements

***Accuracy.*** The robot should read precise values from sensors, and should issue correct and accurate commands to the actuators, so that the movements are performed within acceptable error margins. A violation of this requirement could translate into small defects in the outcome. For example, if the robot is used for welding, a minimal change in how the weld is carried could structurally undermine the workpiece, which in the case of a car body could possibly mean tragic consequences for the end user safety.

***Safety.*** Given their paramount importance, safety requirements are rigidly specified by ISO standards. They can be summarized as: expose sufficient and correct information so that operators can take safe and informed decisions; allow operators to engage emergency procedures; execute emergency procedures quickly and safely.

***Integrity.*** The robot controller should minimize the risk that badly written control logic could result in damage to its physical parts. Violating this requirement means physically damaging the manipulator, or even the controller itself, since several high-voltage electric parts are used by the controller to drive the servo motors.

We consider an attack to be any physical damage to the architecture, or any violation of these requirements—if initiated through a digital vector.

## V. ATTACKING INDUSTRIAL ROBOTS

This section describes industrial-robot-specific attack opportunities to violate the accuracy, safety, and integrity requirements. For each attack, we describe the targeted functional level in the model of Figure 3. Table II summarizes the attacks and the corresponding requirements they violate. Instead, in Section VI-G we discuss how an attacker can leverage *digital* vulnerabilities to carry out these attacks.

### A. Attack 1: Control Loop Alteration

This attack targets the *servo level*. It leverages the fact that, for flexibility and code-reusability purposes, kinematics and configuration parameters are read from a file or defined dynamically at runtime. An attacker able to access a configuration file can modify these parameters.

For the attacker, the most interesting parameters to modify are the ones affecting the robot movements: An extreme parameter modification can completely violate functional and safety requirements.

**Closed Loop Control Parameters Detuning.** A precise control of the position of each link is crucial to ensure that the robot is closely following the desired trajectory, especially over long periods of time. To this purpose, industrial robots adopt closed loop control techniques, such as industry-standard Proportional Integral and Derivative (PID) and Proportional position Integral and proportional Velocity (PIV) control systems, for the angular position of each joint axis.

In general, the aim of a closed loop control system is to make the controlled variable follow a reference signal (set point) as closely as possible. The values of its parameters affect "how well" the controlled variable is able to track the set point, thus influencing the precision of the movement, and the voltage of the servo motors. With a sub-optimal tuning, the controller will only slowly reach the desired position, violating the accuracy requirements. This, in turn, affects the quality of the outcome (i.e., the workpiece can be milled too much, or the welding process can be compromised). Furthermore, wrong parameters may lead to controller instability, causing overshoots over the desired set point. This can result in a violation of safety properties, and cause mechanical stress that could damage the robot.

**Open Loop Control Parameters Detuning.** Speed and position control are usually implemented with additional open-loop actions, employing filters to smooth the signal generated by the closed loop control. This means that any change to the configuration of this part will directly and immediately affect the outputs (position and speed). This could severely amplify resonance effects, violating the integrity requirements of the robot, or cause overshoots of joints, bypassing the safety boundaries.

**Robot Arm and Workpiece Configuration Tampering.** Since a single model of controller must drive different robots, the physical characteristics of the manipulator (i.e., arm and joints) are configurable. This configuration is part of the knowledge model of the robot, and will affect the overall dynamics of the system. The workpiece and the manipulator *are part* of the system, because they have a characteristic weight, shape, and center of gravity and mass, which change over time (e.g., when the workpiece is cut in half, or another manipulator is installed).

Any unexpected modification of these parameters can result, for example, in an amount of applied force exceeding the safety limits, or simply destroying the workpiece or surrounding environment.

In the case of co-bots, which operate with no physical fencing [24], this aspect raises safety concerns, to the point that standards (e.g., ISO TS 15066 [25]) define the maximum force and pressure levels that a co-bot can apply against each relevant part of the human body.

**Safety Limits Tampering.** The control loop parameters comprise the speed limits (e.g., when in manual mode) and the characteristics of the brakes (e.g., minimum activation time). Although safety measures are mechanically actuated, since control loop parameters can be configurable at runtime, the attacker is given an opportunity to bypass safety measures, or to change the precision of the robot's movements.

### *B. Attack 2: User-Perceived Robot State Alteration*

The operator interface must provide timely information at least on the motor state (on/off) and on the operational mode (manual/automatic). Moreover, standards [12] mandate that safety-critical conditions (e.g., restarting a robot from the stop status) require a deliberate user confirmation.

Unfortunately, some of these conditions (as well as the user's acknowledgment) are communicated via software, not through electrical components (e.g., LEDs, buttons). This is the case of current models of co-bots. Thus, the impact of a mere UI-modification attack is remarkable. Altering the UI could hide or change the true robot status, fooling operators into a wrong evaluation of the risk, and, consequently, creating a substantial safety hazard.

### *C. Attack 3: Robot State Alteration*

The attacker can go beyond altering the *perceived* state: Under some conditions, the attacker could alter the *true* state of the robot, while the operator remains unaware. This attack could be combined with other attacks to obtain a greater impact: For instance, a workpiece could be altered without even the controller noticing.

**Motor State.** In some controllers, such as in ABB's YuMi co-bot, the switch between "manual" and "automatic" mode is triggered via a software panel implemented in the teach pendant, not via a hardwired physical switch. As a result, the human operator would trust the robot when in manual mode and operate nearby it, while the attacker could be silently changing the mode of operation to move the robot arm(s) at full speed causing physical harm to the nearby human.

**Software or Wireless Safety Features.** Some vendors implement safety features, such as emergency stop (e-stop) buttons, in software. Worse, modern teach pendants, which of course must include an e-stop button, are wireless (e.g., COMAU's wireless teach pendant).

Therefore, safety features are subject to man-in-the-middle or interface-manipulation attacks. For example, a man-in-the-middle attacker can cause denial of service (i.e., forcefully stopping the robot during normal operation). Moreover, an attacker can disable safety features, thus preventing legitimate users from triggering the e-stop procedure in case of emergency, with clear implications to the safety of the operator.

### *D. Attack 4: Production Logic Tampering*

This attack refers to the *task level*. If the controller does not enforce end-to-end integrity of the task program, an attacker can leverage a file-system or authentication-bypass vulnerability to arbitrarily alter the production logic. For example, the

attacker could insert small defects, trojanize the workpiece, or fully compromise the manufacturing process.

### E. Attack 5: Calibration Parameters Tampering

This attack targets the *sensory processing* and *knowledge model* levels. It is essential for any control system to know the precise axial positions, and to compute the error; thus, the first time a robot is connected to a controller, or after any configuration change, the sensing equipment must be calibrated. Calibration is used to compensate for known measurement errors when triggering servo motors.

The calibration data, initially stored in the sensing equipment, is transmitted to the controller during system boot. Then, the controller uses its local copy of the data.

When the robot is not moving, an attacker can manipulate the calibration parameters on the controller. When the robot starts moving, such manipulation has the effect of forcing a servo motor to move erratically or unexpectedly, because the true error in the measured signal (e.g., joint position) is different from the error that the controller knows. This has the concrete consequence of violating all of the requirements.

If such a malicious manipulation happens *while* the robot is moving, there are two possible outcomes. If the controller does not supervise speed and positions, the outcome is the same of the previous case, with the additional effect that not only the final position of a joint will be affected, but also the maximum speed. Instead, if the controller supervises speed and positions, it can detect unexpected movements, and engage stopping procedures. While the latter case does not result in a violation of any requirement, if the attacker repeatedly triggers such manipulations at "runtime," it can lead to a denial of service attack: The robot will persist in the stop status.

## VI. Case Study

To evaluate the feasibility of the presented attacks for an adversary described by our model, we evaluate the attack surface of a reference robot from a leading vendor (ABB) that implements the architecture described in Section IV.

After presenting an in-depth technical analysis of the deployed robot (Section VI-B), we practically analyze its attack surface and security model (Section VI-C), and we present a set of vulnerabilities that, if exploited, lead to a complete compromise of the controller (Section VI-D and VI-E). As a proof of concept, in Section VI-G we show how to use them to violate safety, integrity, and accuracy requirements, leading to industrial-robot-specific types of attacks.

**Note.** We experimentally verified the presence and the exploitability of the vulnerabilities that we discovered, and promptly disclosed them to the vendor (Appendix B). Where applicable, we include a reference to the vendor's security advisory. While most of the vulnerabilities have been fixed, some are not easily solvable without breaking the boot process. We remark that, even though most controllers share the same industrial standards and have a similar architecture, this does not necessarily means that they share the same software or implementation vulnerabilities. The focal point of our results are not the specific instances of the vulnerabilities that we found, but, rather, their *usage* in complex attacks to industrial robot architectures.

### A. Experimental Setup

Our experimental setup consists of an ABB 6-axis IRB140 industrial robot, capable of carrying a 6 kg payload, equipped with the widely-deployed IRC5 controller (Figure 4) running RobotWare 5.13.1037[1], and the Windows CE-based FlexPendant (teach pendant). The IRB140 must operate in a cage and relies on the default, standard safety measures.

Overall, the total cost of our experimental setup, including the industrial robot, controller, cage, cabling, compressed air piping, and installation fees, is around $75,000, excluding the vendor's 24/7 maintenance service contract.

### B. Technical Analysis of IRB140/IRC5

Figure 4c schematizes the components of the IRC5 controller, with the internal and external data connections. All the internal components are easily reachable since the lock present on the chassis is only meant to prevent accidental electrical shocks and can be easily bypassed with a screwdriver.

**Flex Pendant (FP).** The ABB FlexPendant is an ARM-based system equipped with a touch screen, manufactured by Keba [26], with a pre-installed .NET Compact Framework 3.5. It is designed to run custom applications developed using a vendor-provided .NET-based SDK. The FlexPendant is equipped with safety devices (i.e., emergency stop and dead-man switch), which are electrically connected to the panel board, and ethernet-connected to the main computer.

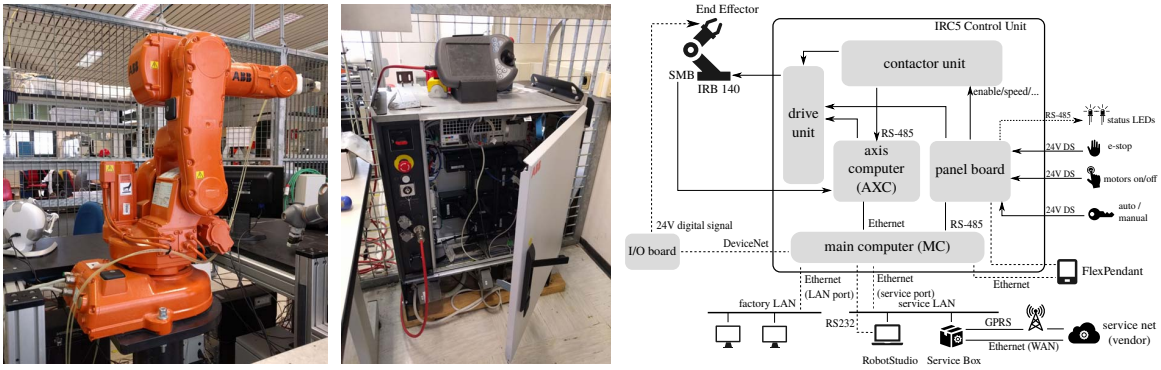
When the robot is in manual mode, the operator can make it move by issuing commands via the teach pendant, and must keep the teach pendant's dead-man switch pressed at all times while the robot is moving. Instead, in automatic mode, the robot runs previously stored programs in an unattended fashion.

**Main Computer (MC).** The main computer is based on the Intel x86 architecture, and runs the VxWorks 5.5.1 RTOS. Being responsible for the task, action, and primitive level control, it orchestrates the execution of tasks and coordinates the controller's components: It interprets the task program code written in ABB's RAPID language, manages the execution of tasks, chooses the best control strategy, applies forward and inverse kinematics, and implements the path planning strategy.

**Axis Computer (AXC).** The AXC—a PowerPC-based board running the VxWorks 5.4.2 RTOS—implements the "servo level" of the abstract control system in Figure 3. This computer controls the servo motors that operate the joints through the drive and contactor units. The contactor unit is a switching device that controls the status of the motors (on/off), whereas the drive unit provides power to the motors of the manipulator.

In addition, the AXC feeds back to the MC any data needed in the planning phase (e.g., position and revolution counters).

[1]The controller we used only supports RobotWare 5.x. However, we verified with ABB and by manual reverse engineering that our findings also apply to the latest version of RobotWare 6.x at the time of running the experiments.




(a) Caged arm and end effector. (b) Controller and teach pendant. (c) Dashed lines are optional connections.

Figure 4. Experimental setup: ABB IRB140 industrial robot (left), IRC5 controller and teach pendant (center), and schematics (right).

Finally, to estimate the error and complete the control loop that generates the drive signals, it acquires from sensors the position and status of each servo motor.

**Panel Board (PB).** The panel board, or safety board, is mostly based on discrete integrated circuits and relays that block the robot when safety-critical conditions occur. It implements the safety requirements mandated by the standards (ISO 12100 [27]): Therefore, it is not programmable by the user. The PB is connected to the MC via a 1 MHz RS485 single-duplex connection, used to send a heartbeat packet containing the current status of the robot: motors on/off, emergency stop status, operating mode (manual reduced speed, manual full speed, automatic).

The MC periodically sends a keepalive signal to the PB, and will interrupt the MC whenever a change in the operating state is detected. For efficiency, this is implemented through an FPGA wired to the MC's PCI board.

**Network Connections.** The MC is the most exposed component of the robot: It communicates, through ethernet connections, with any external devices and with the teach pendant. Internally, the AXC and MC also communicate through a standard ethernet connection. The controller, and ultimately the MC, can be connected to a local network through the LAN ethernet port, whereas a secondary ethernet port (the "service port") enables direct connection with a client host for maintenance and programming purposes. Although, by default, any host connected to the service port is assigned an IP address on a separate subnet, the MC bridges the service port to the FlexPendant's ethernet connection.

Furthermore, on the *external* side of the chassis, there are a host-mode USB port directly connected to the MC, the main power switch, the emergency stop button, and a keyed switch to change the operating mode (manual or automatic), connected to the panel board. Notably, the USB port can mount mass-storage devices.

### C. Attack Surface and Security Model

Modern industrial robots expose a considerable attack surface for both network and physical attackers. The main entry points are summarized in Table III. We will focus on the network attack surface, a sub-set of the physical (local) surface. Both the MC and the FP expose a number of network services that are essential for the operation of the robot. There are standard services such as FTP, used to share files and system information between the robot and the internal network, and custom services. The RobAPI [28] is a custom service that offers the most extensive network attack surface: It is a complex and partially authenticated API that both the FP and any host connected via the service or LAN port can use to "talk" to the MC. RobAPI clients are expected to use the RobotStudio suite, but of course valid RobAPI messages can be crafted and encapsulated inside standard TCP/IP frames. Moreover, the MC exposes a UDP-based discovery service, used by the FlexPendant and by RobotStudio to automatically discover robot controllers on the network. The FP instead uses broadcast UDP packages to send debug information and messages. The AXC communicates with the MC via a proprietary protocol on top of Ethernet. This protocol is uninteresting for our purposes since it is not directly exposed.

Table III
ATTACK SURFACE BY CHANNEL

| Access | Channel |
|---|---|
| Physical | - USB port<br>- Industrial bus access via, e.g., after-market end effectors<br>- Ethernet: LAN service port<br>- Direct access to internal devices (e.g., axis computer) |
| Local | - LAN port |
| Remote | - WAN access to (un-firewalled) LAN port<br>- WAN access to remote service facilities, i.e., service box |
| Wireless | - Wireless (e.g., GSM) access to remote service facilities |

The MC is the most sensitive entry point, as it exposes various services to the network, and gaining unauthorized access to this component leads to a complete compromise of the controller. It performs sensitive operations on its own, and the communication between internal components of the robot is implicitly trusted. For this reason, we dedicate a separate section to its security analysis: Section VI-D.

**User Authentication System.** According to the documenta-

tion, the User Authentication System (UAS) is optional, and in place "for protecting data and functionality from unauthorized use" [29]. When the UAS is enabled, the controller enforces access control rules according to a simple role-based model. When logging in through a client device (e.g., FlexPendant, or a computer running RobotStudio), a username and password are required. A user belongs to one or more *groups*; a group specifies multiple *grants* that define the actions users are allowed to perform on the controller. Grants are divided into *application grants* and *controller grants*: The controller grants are enforced on the main computer, whereas the application grants are validated by specific applications running on the client device without involving the controller. For example, the grant that allows one to manage the UAS settings, or the grant that allows a user to read or write files are controller grants; the grant that controls access to a specific menu on the FlexPendant is an application grant. An API also allows developers to use the UAS to perform access control in custom FlexPendant applications.

**Boot Process.** When the controller is powered on, the MC boots, and exposes an FTP service to allow the AXC and the FP to download their firmware image. The AXC will use hard-coded credentials to connect to the MC, while the FP will use credentials that have been stored in the Windows registry (set to default values). The FP will be responsible for retrieving the correct version of the image out of many present on the MC. Any firmware or application file coming from the FTP server is implicitly trusted.

### *D. Security Analysis of the Main Computer*

**Unsecured Network Surface and Command Injection.** Network-exposed services are an important attack surface. For example, an attacker can abuse the FTP-exposed file system to read and modify configuration and program source files and, ultimately, to control the actions that the robot executes.

Exposing the file system over the network may lead to deeper consequences. In the VxWorks RTOS, filesystem operations are used to access devices, which are mounted in a single directory hierarchy. On the MC, a custom device driver is mounted at `/command`. When reading files in the `/command` directory, the device driver returns information about the MC. The FlexPendant's boot loader, for instance, uses this mechanism to read the main computer's environment variables, synchronize the clock, and retrieve startup information. Any file written over FTP to the path `/command/command` or `/command/command.cmd` is interpreted as a script: Each line of the script must contain a command (out of a set defined by the driver), a white space, and a parameter. The remote service box uses this functionality to automatically configure itself.

Interestingly, the command `shell` executes an arbitrary VxWorks symbol passed as a parameter (ABBVU-DMRO-124642, fixed as of RobotWare 5.15.12, 6.03.02 and 5.61.07): This feature can be leveraged by an attacker to bypass the UAS or to execute functions in an unintended way. For example, by writing a file containing the line `shell reboot`, the main computer performs a warm restart; with the command `shell uas_disable`, the user authentication system is temporarily disabled (and, as a side effect, the system reboots).

**Weak Authentication.** The UAS protects the controller from unauthorized FTP and RobAPI access. We found that, due to implementation flaws, an attacker can bypass it.

First, to allow the FP to retrieve configuration information, authentication is disabled during the system boot: During this phase, the default static credentials can be used to access the shared file system. In fact, the FlexPendant boot loader logs in with a specific username and a hard-coded password. If the attacker knows when a reboot takes place or is able to trigger one, the controller can be remotely accessed using those credentials.

Second, the UAS comes with a default user, without password, that cannot be changed or removed. Although it is possible to revoke the sensitive permissions granted to this user, the documentation explicitly warns that there is a risk of being locked out by changing the group membership of the default user, without fully explaining the security implications of leaving meaningful grants assigned to it.

Third, we found that a specific user, used by the service box to exchange data with the main computer, has a set of hard-coded credentials that are embedded in the MC's firmware and cannot be changed (ABBVU-DMRO-124644, fixed as of RobotWare 6.01). Although this user can only access FTP paths related to the `/command` device driver, this makes the aforementioned command injection vulnerability exploitable without authentication.

**Naïve Cryptography.** An attacker with read-only filesystem access is able to tamper with the UAS configuration, changing the privileges of existing accounts and changing or retrieving the password of all users. The UAS configuration (including the plain-text passwords of all users) is stored in an XML file obfuscated through a bit-wise XOR operation with a random key; as the key is stored at the beginning of the obfuscated file itself, the obfuscation is completely useless. The vendor considers this behavior "by design," and asserts that the UAS's purpose is more related to safety (assign users different roles and prevent users from making mistakes) than to security.

More generally, encryption schemes are used to safeguard the integrity of some specific and safety-critical configuration files, such as the ones containing sensitive control loop parameters. We found such schemes to be weak obfuscation/integrity mechanisms rather than proper encryption: keys are derived from the file name and, in some cases, part of the file content. By reverse engineering the controller firmware, we found all the information needed to reconstruct the encryption keys: An attacker who can access a firmware update, or who has file system access, is able to read and modify safety- and accuracy-critical configuration files.

**Memory Corruption.** A proprietary protocol, RobAPI, is used to access a set of services exposed by the controller:

changing the configuration, moving the robot, obtaining information about the current task, and controlling the task program execution. RobAPI consists of messages that are exchanged synchronously (i.e., request-response) and asynchronously (i.e., through event subscriptions). Most of the RobAPI functionality is authenticated through the UAS. Upon authentication (once per TCP connection), the MC returns a user ID that the client sends as part of all the messages. The exposed RobAPI functionality is also divided into a set of *domains*. For example, the `CONTROLLER` domain gives access to general settings, and the `RAPID` domain controls the execution of RAPID programs.

We found an exploitable memory error in the code that receives RobAPI requests for the `DHROOT` domain. The error is a textbook stack-based buffer overflow in the function `DHROOT_SET_REQ`. More precisely, when the command `DomainHookTest` is invoked, the property field of the request string is copied to a buffer using the `strcpy()` function, without checking whether the user-provided string fits in the statically allocated space. The specific vulnerable endpoint does not require authentication (ABBVU-DMRO-124645, fixed in release 5.15.12, 6.03.02 and 5.61.07).

Similarly, we were able to find other stack-based buffer overflows exploitable in a similar way in the parsing routines of the `/command` endpoint (ABBVU-DMRO-128238, fixed in releases 5.15.13, 6.04 and 5.61.07). We experimentally validated that an attacker can exploit these vulnerabilities to obtain remote arbitrary privileged code execution on the MC. As the MC firmware does not have mitigation mechanisms against the exploitation of memory corruption errors, and there is no privilege separation between processes or between user and kernel land, exploiting memory corruption is trivial.

### *E. Security Analysis of the FlexPendant*

**Missing Code Signing.** The boot image that the FP downloads from the MC is not signed, and we verified that it can be easily modified by an attacker who is able to reverse engineer the file format. We used this issue to implement the UI attack depicted in Figure 6.

We were able to reverse engineer the image format, which is a ZIP-compressed binary containing the file name, size and content. In Section VI-G we show that an attacker can use this vulnerability to execute arbitrary code on the FP, by tampering with the data exchanged between MC and FP (e.g., by previously compromising the MC or—if the attacker is local—through network attacks such as ARP spoofing).

**Memory Corruption.** We found a memory error in the executable *TpsStart.exe*, executed during the FP's startup process. If an attacker is able to tamper with a specific file when it is retrieved from the MC (`/command/Timestamp`), they can trigger a stack-based buffer overflow by making sure that the retrieved file name is longer than 512 bytes (ABBVU-DMRO-124645, fixed in release 5.15.12, 6.03.02 and 5.61.07). Exploiting this vulnerability can block the FlexPendant boot, resulting in a denial of service, which in turn offers to the attacker the opportunity to expect a reboot—and thus launch some of the previously mentioned exploits.

**Poor Runtime Isolation.** Programmers can develop applications that run on the FlexPendant in two ways: using the RAPID programming language with the ScreenMaker utility, or using the FlexPendant SDK.

The RAPID language allows one to control robot positions, to perform I/O processing, and to use network sockets. However, it does not allow one to manipulate processes or call native APIs, and it provides only limited file system access.

The FlexPendant SDK allows richer access, and is thus more interesting to an attacker. The SDK is composed of .NET Compact Framework 3.5 assemblies, Visual Studio templates and a *compliance tool* that generates a DLL library used by the Application Host Framework to load the assembly as part of the "graphical teach pendant unit". We found out that the compliance tool, as part of the checks it performs, tries to ensure that the code does not use a set of forbidden operations: reflection capabilities and the use of some specific .NET namespaces that allow access to raw filesystem and RobAPI capabilities intended to be used only in vendor-provided libraries, not in custom applications. The tool performs this check by means of regular expressions and by using a disassembler for the .NET bytecode (ILDasm).

Alas, the version of the compliance tool provided with the SDK does not enforce these restrictions at all. It is also evident that an attacker could bypass this by simply modifying the compliance tool itself: There is no way to perform these security checks on the programmer's side in a completely safe manner.

In addition, we found some limitations in the operations allowed with the FlexPendant SDK: Access to the filesystem is naïvely regulated through a blacklisting process implemented in the `FileSystemDomain` class of the `ABB.Robot.Controllers` namespace. This blacklist denies access to the `CTRLROOT` and `INTERNAL` directories. Using assemblies part of the namespace `ABB.Robotics.Dcl`, which should be blacklisted by the compliance tool, we can obtain full unrestricted access to the shared file system, and execute any operation allowed by the full RobAPI (obeying only the UAS privileges—when applied). This also allows reflashing the memory of the pendant, resulting in full compromise of the TPU.

In some versions of the IRC5 controller (e.g., the IRC5 Compact used in ABB's YuMi), the switch between automatic and manual mode is performed entirely via software, and not delegated to a physical key wired to an electrical circuit and a speed limiting line. Alarmingly, the YuMi is a co-bot, which operates nearby the operator, with no cage required. Since having full privileged access to the teach pendant means that an attacker is able to patch system-level .NET assemblies, or even to overwrite the firmware with a customized malicious one, this also allows an attacker to abuse software-defined switches to toggle the operating mode without user intervention, posing significant safety risks to the operators.

Furthermore, although Microsoft dropped support for the .NET Compact Framework 3.5, the vendor will continue to provide the FlexPendant SDK on future pendants for backward compatibility reasons—meaning that the FP will continue running an old, unsupported, and potentially vulnerable version of the .NET framework [30].

### F. Attack: Controller Exploitation

We will now describe a multi-step exploitation path, comprising several of the presented vulnerabilities. This can be leveraged to seize complete control by the most general of the adversaries: a remote network attacker who has no knowledge of valid UAS credentials, and who can connect to the RobAPI and FTP services.

1) **Main Computer Compromise.** We used the FTP static credentials we discovered to access the `/command` driver, and, alternatively, exploited the memory errors found in the RobAPI to gain initial access.
2) **Authentication Bypass.** We temporarily disabled the UAS and triggered a full system reboot. To this end, we invoked `uas_disable` via the shell function of the FTP `/command`, and, alternatively, via the exploitation of the remote code execution vulnerability (e.g., the RobAPI memory error).
3) **Payload Upload.** With full FTP access without credentials, we permanently disabled UAS by editing the obfuscated configuration file, and then uploaded custom, malicious, .NET DLL (or RAPID code) to the running system directory on the controller.
4) **Persistent Access** We triggered another reboot via FTP (`/command` shell function), causing the FlexPendant to auto-execute the uploaded malicious .NET libraries—or the MC to load the malicious RAPID files.

In the FlexPendant context, due to the lack of proper sandboxing, the attacker has now complete access to the OS resources, and any arbitrary remote code execution vulnerability in the RobAPI would imply unrestricted control of the MC. Another possibility is to leverage full unrestricted access to the FTP to upload a maliciously crafted RobotWare image, allowing the attacker to compromise all the other components, because the firmware images are not signed.

**Command & Control.** Although not essential, we used the FTP-accessible file system as a simple command & control server to exchange data with the compromised FlexPendant, without needing a direct network connection to it. Alternatively, an attacker can load a RAPID module that is executed on the MC and acts as a relay between the FP and the remote attacker. We can consider that, at this point, our attacker has completely compromised the robot controller: We describe the attacks' implementation starting from this assumption.

### G. Attack: Robot Exploitation

We can now explain how we implemented robot-specific attacks to show that an adversary can affect the production chain after compromising the robot controller.

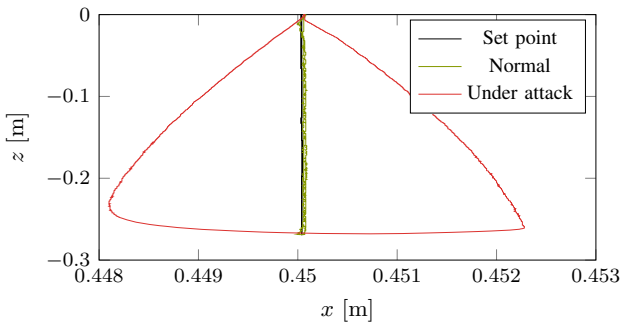



Figure 5. (Section VI-G1: Accuracy Violation) End effector 2D trajectory when the robot is programmed to move south/north on a straight line.

As discussed in Section VIII, some attacks presented in Section V can have a disruptive impact, up to the entire cost of the deployment or the violation of safety policies and regulations. When unable to implement an attack entirely, we discussed our implementation with a domain expert, who confirmed its feasibility and its potential destructive effects.

*1) Accuracy Violation:* The robot we analyzed uses a closed loop controller to control the joint position. For each joint, a PID controller ensures that its angular position follows, as closely as possible, the reference trajectory needed to complete the task. As mentioned in Section V-A, if an attacker is able to "detune" the controller, they can reduce the accuracy of the movement and, ultimately, impair the precision of the system.

The PID parameters are stored in a robot-specific configuration file on the MC file system[2]; this file is naïvely obfuscated as described in Section VI-D.

To precisely measure the trajectory of the end effector under nominal and attack conditions, we decoded such data from the RS232 service port and real-time ethernet, by enabling a debug functionality in the MC, which allowed us to collect the coordinates of the position at fast sampling intervals. While doing so, we did not interfere with the attacker's capabilities.

By leveraging the remote code execution vulnerability, we modified the control-loop configuration files, which are naïvely obfuscated and, thus, easily modifiable. In particular, we changed the proportional gain of the first joint's PID controller, setting it to 50% of its original value. Then, we programmed the robot to perform a straight horizontal movement. Figure 5 shows the trajectory of the end effector projected on the horizontal plane, which is notably altered. Although the maximum difference between the position under normal conditions and under attack is small (less than 2 mm), according to the specific machining that the robot is performing, it can be enough to destroy the workpiece.

*2) Safety Violation:* To violate safety requirements, we used the User-Perceived Robot State Alteration approach (Section V-B) to trick the operator into thinking that the robot is in manual/motor-off mode, whereas it is in auto/motor-on mode. Recall that in auto/motor-on mode, an attacker is able to load a

[2]As an example, the configuration file for the IRB140 robot is located in the `robotware` directory under `robots/irb1_140/irbcfg/sec_140_0.81_5_typea_1.cfg.enc`.

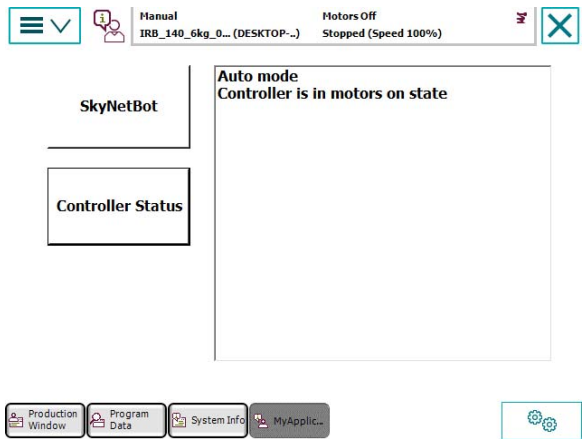



Figure 6. (Section VI-G2: Safety Violation) Modified teach pendant showing motors off/manual mode while in auto/motor on.

program task that pilots the robot, while the operator believes that the robot is in manual/motor-off mode, and thus it is safe to get close to it.

We implemented this attack by leveraging the lack of code signing in the FP firmware, which we reverse engineered to find the routines that implement the user interface. We modified the strings that are output on the display, to show false information to the operator, and re-packaged the binary firmware. Then we used the authentication bypass on the FTP server to load the binary on the MC's file system, and we waited for the next reboot that had the effect of loading the malicious firmware onto the teach pendant. As an alternative, we could have leveraged the command-injection vulnerability to force the MC to reboot remotely. Figure 6 shows the modified UI.

Although we did not have access to the YuMi co-bot, we noticed that it shares the same software as the IRC5 controller with only insubstantial modifications. By reverse engineering the .NET assemblies of the YuMi's FlexPendant, we found evidence suggesting that the mode of operation can be changed via software as described in Section V-C.

*3) Integrity Violation:* It is possible to violate integrity properties through the control-loop alteration approach (Section V-A), and the calibration-parameters tampering approach (Section V-E). We wanted to overshoot the joints in order to make the robot collapse on itself, and to force the servo motors beyond its physical, structural limits; this attack is costly and potentially destructive, because its goal is to damage the robot.

Alternatively, an attacker could use the robot state alteration approach (Section V-C) to repeatedly and abruptly start and stop a servo motor, causing wear to the electro-mechanical components, the brakes, and the servo motor itself.

We implemented the software stages of this attack using two vulnerabilities: first, we created valid encrypted configuration files containing arbitrary calibration parameters; second, we modified them on the controller, exploiting the remote code execution bug in the MC.

At this point, instead of starting the robot, we showed the modified configuration files to a domain expert (a lab technician who normally operates the robot), who confirmed the destructive effects on the robot. Additionally, we checked the effect of such configuration changes on the sections of the code that implement the control and supervision routines, which we reverse engineered manually. The only integrity checks that we found in the speed and position supervision routines can be easily bypassed modifying the MC's control logic through the use of any of the vulnerabilities we found.

## VII. Discussion

### A. Cyber Security and Safety Standards

Industrial robots standards emphasize safety requirements. For example [12], [31], they define performance requirements, stopping functions, e-stop features, required pendant controls, and speed bounds. Unfortunately, none of the standards explicitly account for cyber-security threats: Although some of them have *mild* security implications, they do not explicitly account for adversarial control during risk assessment.

Instead, cyber security issues in ICS [32] and automotive [33] have received better attention from standardization bodies. We hope that our work will serve as motivation to develop similar standards for industrial robots.

### B. Security Measures and Challenges

The already hard task of designing a secure architecture without sacrificing functionality is exacerbated by challenges in providing timely security updates, a well-known problem in embedded systems and ICS [34]. Compared with other ICT systems, industrial robots have a very long lifetime, which increases the burden on vendors to support several deployed devices, leading to so-called "forever-day" vulnerabilities (i.e., well known vulnerabilities that are never patched). Moreover, there is friction to adding system-level hardening features, which renders the exploitation of vulnerabilities more probable than on mainstream OSs. This is amplified by the so-called "patching problem" well known to ICS security practitioners: As industrial robots are critical to productivity, customers may be worried about potential downtime or regressions caused by software updates, and refrain from timely patching their systems.

We overview the main challenges that arise when applying even textbook-level security practices in the industrial robots domain.

*1) Human interaction:* Human intervention can modify the outcome of the attacks we presented or stop them. For example, if emergency stops are implemented by means of electro-mechanical switches (i.e., not via software), an operator who spots an abnormal behavior can promptly halt the robot.

However, some attacks can alter the user-perceived robot state (section V-B), confusing the operator, whose reaction time will be too slow to counteract a quick and unexpected movement of the robot. Furthermore, attacks can be more effective when used in a stealthy way, e.g., by introducing small defects or stealing intellectual property.

*2) Attack Detection and System Hardening:* Effective and readily applicable attack-detection approaches are needed to provide a short- to medium-solution for threat mitigation.

***Attack Detection (Short Term).*** Research and industry efforts should provide short-term solutions to mitigate the impact of vulnerabilities. For example, detection and correction of anomalies must be explored, rather than focusing on access control techniques only. As for automotive systems [5], prevention measures may interfere with the production chain and induce downtime: thus, a delicate balance between detection and prevention should be considered.

***System Hardening (Short/Medium Term).*** As it is hard to patch all vulnerabilities in complex software, medium- to long-term solutions include system hardening to make reliable exploitation more expensive. Implementing these techniques in legacy embedded platforms is challenging; they may require hardware support and design changes.

Notably, it is far easier to exploit memory corruption vulnerabilities in the robot we analyzed than in mainstream OSs. Industrial robots, like many embedded systems, employ real-time operating systems (RTOSs) with poor or no hardening features. Surprisingly, although RTOSs are used in critical tasks, most research in this area is focused on determinism, efficiency and safety, rather than on system security.

An effective, short-term way to harden a RTOS is by enabling OS- and compiler-based mitigations, such as ASLR, DEP, and canaries. This set of functionality is common, but is not always supported in embedded systems.

Another effective containment measure is to enforce privilege separation at the OS level, and to require physical separation of critical functionality across different subsystems. The challenging part here is the trade-off between security and real-time requirements. For instance, the VxWorks 5.x OS used in ABB's RobotWare 5.x executes all code in kernel mode and is built as a monolithic ELF binary. Although VxWorks 6 supports user- and kernel-mode demarcation since 2004 for platforms having an MMU (the so-called real-time process model), this functionality is not used even in newer versions of RobotWare (over twelve years later).

This reasoning also applies also to functionality aimed at executing custom code: Even though the flexibility of industrial robots requires the execution of customized software, they do not need access to any functionality of the underlying OS. Custom applications and robot programs can be limited to a sandboxed environment, following the principle of least privilege. For example, Microsoft .NET's Application Domains [35] can be used as a light sandboxing mechanism in .NET-based embedded systems such as the teach pendant.

*3) Software Design and Deployment Challenges:* We identified some steps that vendors can adopt to reduce the impact of security issues. Although well-known in the security community, applying them in this domain is challenging, due to time-consuming patching processes, as well as non-trivial changes in the controller design.

***Program Protection (Short Term).*** Removing or making it easy for the users to disable detailed debug outputs and symbols can play a key role in increasing the cost of an attack, especially for casual attackers. In the system we analyzed, the executable binary of the main computer was not stripped, and detailed debug output was readily available from the serial console, which eased reverse engineering and exploit development. We also observed that the FlexPendant broadcasts detailed debug information through UDP packets, making it accessible to a physical attacker connected to the service port of the main computer.

***Secure Software Development Lifecycle (Long Term).*** In general, enforcing secure software engineering practices can improve code robustness and harden the underlying platform. Such practices range from forbidding the use of unsafe C library functions (e.g., `strcpy`, `strcat`, `sprintf`), to using static-analysis tools to find potentially problematic code regions and unsafe implementation patterns. These practices must be a part of a comprehensive process that takes security into account during the whole software development lifecycle. Unfortunately, the fact that we have found textbook vulnerabilities in one of the most widely deployed robot controllers is an indicator that not even basic static checking was in place.

***Secret Management (Medium Term).*** In our case study we observed frequent use of (i) static, wired-in credentials, shared among the devices of the same model, (ii) obfuscation of passwords (as opposed to hashing and salting), and (iii) naïve "encryption" of configuration files. These measures create a false sense of security. For example, a superficial review of the software source code may imply that certain functionality is strongly authenticated, whereas it is instead accessible with a password readily available in the firmware executable.

***Component Interconnection Hardening (Medium Term).*** In the domain of industrial robots (as well as the ICS and automotive ones), the threat model assumes the internal network to be trusted. This assumption is not realistic in scenarios where a component is compromised: Even if such a component is not critical to the operation of the system, it may grant the attacker access to the trusted network. An effective hardening measure is to move toward an Industry-4.0-compliant threat model, without a trusted internal and factory network, where the boundary between internal and external is dynamic. This implies, for instance, appropriately filtering inputs coming from connected components as if they were coming from an untrusted party, and taking into account eavesdropping and tampering of messages.

For example, some UI alteration attacks (Section V-B) can be mitigated by not trusting the teach pendant: A switch hardwired to the controller can be used to require the operator's acknowledgment before critical state changes.

***Code and Configuration Signing (Medium Term).*** An effective system-level mitigation is the implementation of code signing mechanisms with strong authentication.

Broadly speaking, we can distinguish between three types of executable code: vendor-provided firmware, program task code, and custom software developed on top of a vendor-provided software development kit (SDK). It is expected that only the vendor can develop and run updated firmware for the

robot components. For custom code, the customer must be able to sign code for its own robots (but not for other customers' ones): This way, an attacker who can upload arbitrary files to the robot file system would not gain arbitrary code execution.

Solving the problem on a global scale is challenging due to the trade-off between safety and security, and development and deployment time and effort. A possible way to tackle it is using a PKI with the vendor as a certification authority, issuing per-customer certificates.

Fortunately, commercial embedded OSs and hardware platforms are moving toward supporting secure boot capabilities (e.g., the Security Profiles in VxWorks 7), which would allow implementing a full code-signing chain.

A mechanism to keep the flexibility of the teach pendant programming model intact, if sandboxing mechanisms are in place to limit the privileges of user-provided program task code, is to enforce the code signing policy only when the robot is running in automatic mode. Before switching to automatic mode, the code can be signed offline or even online (e.g., via smart-card devices connected to the teach pendant).

In the system we analyzed, we found various "encrypted" configuration files. It is unclear if the intended use of encryption for these configuration files is to avoid casual tampering with the configuration or if the developers had a more complex threat model in mind. However, these configuration files control safety parameters and signal routings, and are critical for the operation of the robot; if this is the case, requiring that the configuration files are signed (with the same infrastructure in place for custom code) will address this problem effectively.

## VIII. Limitations

***Cost of Exploit Testing.*** The high cost and limited availability of industrial robotics equipment affected the depth of our analysis: we could not perform experiments carrying a substantial risk of permanently breaking electro-mechanical components of the robot (e.g., drive the robot after reaching its physical operating limits). We were bound to respect several security and safety regulations, and not authorized to make such experiments whenever the exact outcome could not be forecast with good accuracy. For example, we could not perform experiments involving introducing excessive controller instability, or where the outcome depended on specific features of the controller that we could not simulate.

***Generality.*** We restricted our case study analysis to standard features of the ABB IRB140/IRC5 robot/controller, without considering optional equipment that could increase the attack surface. Thus, our analysis is conservative. For example, the robot controller supports an optional I/O board based on DeviceNet FieldBus [36], which are known to be insecure [4], [5]. This opens an interesting attack scenario, which we defer to future research, where compromised after-market end effectors can gain unauthorized access to the robot.

***Survey.*** While we think that the survey results are interesting by themselves, a survey targeting more participants is needed to make results statistically significant.

## IX. Conclusions

This paper represents the first step in a broader exploration of security issues in the industrial robotics ecosystem.

We explored, theoretically and experimentally, the challenges and impacts of the security of modern industrial robots. We built an attacker model, and showed how an attacker can compromise a robot controller and gain full control of the robot, altering the production process. We explored the potential impacts of such attacks and experimentally evaluated the resilience of a widespread model of industrial robot (representative of a de facto standard architecture) against cyber attacks. We then discussed the domain-specific barriers that make smooth adoption of countermeasures a challenging task.

Interesting future research directions include exploring multi-robot deployments, co-bots, and the safety and security implications of the adoption of wireless connections. Also, an improved survey would produce statistically significant results. We definitely plan to analyze controllers from other vendors, to further confirm the generality of our approach.

## Acknowledgments

We wish to thank all of the reviewers of previous versions of this paper for suggesting critical improvements. This project has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement nr. 690972; and from the "FilieraSicura" project, funded by Cisco Systems Inc. and Leonardo SpA.

## References

[1] I. F. R. Carsten Heer, "Survey: 1.3 million industrial robots to enter service by 018," 2016. [Online]. Available: http://www.ifr.org/index.php?id=59&df=2016FEB_Press_Release_IFR_Robot_density_by_region_EN_QS.pdf

[2] D. Zuehlke, "Smartfactory - towards a factory-of-things," *Annual Reviews in Control*, vol. 34, no. 1, pp. 129–138, 2010.

[3] A.-R. Sadeghi, C. Wachsmann, and M. Waidner, "Security and privacy challenges in industrial internet of things," in *Proceedings of the 52nd Annual Design Automation Conference*. ACM, 2015, p. 54.

[4] S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham, S. Savage, K. Koscher, A. Czeskis, F. Roesner, and T. Kohno, "Comprehensive experimental analyses of automotive attack surfaces," in *Proceedings of the 20th USENIX Security Symp.* USENIX Association, 2011.

[5] K. Koscher, A. Czeskis, F. Roesner, S. Patel, T. Kohno, S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham, and S. Savage, "Experimental security analysis of a modern automobile," in *Proceedings of the 2010 IEEE Symp. on Security and Privacy*. IEEE Computer Society, 2010, pp. 447–462.

[6] M. Brunner, H. Hofinger, C. Krauß, C. Roblee, P. Schoo, and S. Todt, "Infiltrating critical infrastructures with next-generation attacks," *Fraunhofer Institute for Secure Information Technology (SIT), Munich*, 2010.

[7] "Nccic/ics-cert year in review 2015," 2015. [Online]. Available: https://ics-cert.us-cert.gov/sites/default/files/Annual_Reports/Year_in_Review_FY2015_Final_S508C.pdf

[8] C. Nobile, "Robots vulnerable to hacking," 2012. [Online]. Available: http://www.roboticsbusinessreview.com/article/robots_vulnerable_to_hacking/

[9] A. Y. Javaid, W. Sun, V. K. Devabhaktuni, and M. Alam, "Cyber security threat analysis and modeling of an unmanned aerial vehicle system," in *Homeland Security (HST), 2012 IEEE Conference on Technologies for*, Nov 2012, pp. 585–590.

[10] T. Bonaci, J. Herron, T. Yusuf, J. Yan, T. Kohno, and H. J. Chizeck, "To make a robot secure: an experimental analysis of cyber security threats against teleoperated surgical robots," *arXiv preprint arXiv:1504.04339*, 2015.

[11] S. Morante, J. G. Victores, and C. Balaguer, "Cryptobotics: Why robots need cyber safety," *Frontiers in Robotics and AI*, vol. 2, no. 23, 2015. [Online]. Available: http://www.frontiersin.org/humanoid_robotics/10.3389/frobt.2015.00023/full
[12] ISO, "10218-2: 2011: Robots and robotic devices–safety requirements for industrial robots–part 2: Robot systems and integration," *Geneva, Switzerland: International Organization for Standardization*, 2011.
[13] Microsoft, "Kuka creates a connected factory with iot," 2016. [Online]. Available: https://www.microsoft.com/en-us/cloud-platform/customer-stories-kuka-robotics
[14] ABB Robotics, "Robot web services," 2015. [Online]. Available: http://developercenter.robotstudio.com/Index.aspx?DevCenter=Robot_Web_Services
[15] J. Lambrecht, M. Chemnitz, and J. Krüger, "iphone industrial robot control - kuka kr 6," 2011. [Online]. Available: https://www.youtube.com/watch?v=yFi7UL70zTo
[16] ——, "Control layer for multi-vendor industrial robot interaction providing integration of supervisory process control and multifunctional control units," in *2011 IEEE Conference on Technologies for Practical Robot Applications*, April 2011, pp. 115–120.
[17] J. R. Hagerty, "New robots designed to be more agile and work next to humans: Abb introduces the yumi robot at a trade fair in germany," *Wall Street Journal*, vol. 13, 2015.
[18] L. Cruz, "Digitization and iot reduce production downtime," https://newsroom.cisco.com/feature-content?type=webcontent&articleId=1764957, 2016.
[19] M. Tischer, Z. Durumeric, S. Foster, S. Duan, A. Mori, E. Bursztein, and M. Bailey, "Users really do plug in usb drives they find," 2016. [Online]. Available: https://cdn.elie.net/publications/users-really-do-plug-in-USB-drives-they-find.pdf
[20] M. Rocchetto and N. O. Tippenhauer, "On attacker models and profiles for cyber-physical systems," in *European Symposium on Research in Computer Security*. Springer, 2016, pp. 427–449.
[21] S. Moon and G. S. Virk, "Survey on iso standards for industrial and service robots," in *ICCAS-SICE, 2009*, Aug 2009, pp. 1878–1881.
[22] International Organization for Standardization, *Robots and robotic devices - Vocabulary*, ser. International standard; ISO 8373. International Organization for Standardization, 2012.
[23] B. Siciliano, L. Sciavicco, L. Villani, and G. Oriolo, *Robotics: modelling, planning and control*. Springer Science & Business Media, 2010.
[24] I. Bonev, "Should we fence the arms of universal robots?" http://coro.etsmtl.ca/blog/?p=299, 2014.
[25] ISO, "Ts 15066: 2016: Robots and robotic devices–collaborative robots," *Geneva, Switzerland: International Organization for Standardization, draft*.
[26] Keba, "Keba - automation by innovation," 2016. [Online]. Available: http://www.keba.com/
[27] ISO, "12100: 2010," *Safety of machinery — General principles for design — Risk assessment and risk reduction*, vol. 1, 2010.
[28] ABB Robotics, *Application manual: Robot Application Builder, RobotWare 5.0*. ABB Robotics, 2009.
[29] ——, *Operating manual - Trouble shooting, IRC5*. ABB Robotics, 2010.
[30] ABB Forums, "Information to all users of flexpendant sdk," 2014. [Online]. Available: https://forums.robotstudio.com/discussion/8080/information-to-all-users-of-flexpendant-sdk
[31] ISO, "Iso 10218-1: 2011: Robots and robotic devices–safety requirements for industrial robots–part 1: Robots," *Geneva, Switzerland: International Organization for Standardization*, 2011.
[32] 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.
[33] SAE, "J3061: 2016," *J3061 Cybersecurity Guidebook for Cyber-Physical Vehicle*, 2016.
[34] D. Papp, Z. Ma, and L. Buttyan, "Embedded systems security: Threats, vulnerabilities, and attack taxonomy," in *Privacy, Security and Trust (PST), 2015 13th Annual Conference on*, July 2015, pp. 145–152.
[35] Microsoft, "Application domains overview," 2016. [Online]. Available: https://msdn.microsoft.com/en-us/library/2bh4z9hs(v=vs.90).aspx
[36] ABB Robotics, *Application Manual: DeviceNet, Robot Controller, RobotWare 5.0*. ABB Robotics, 2004.
[37] R. A. Hirschfeld, F. Aghazadeh, and R. C. Chapleski, "Survey of robot safety in industry," *International Journal of Human Factors in Manufacturing*, vol. 3, no. 4, pp. 369–379, 1993. [Online]. Available: http://dx.doi.org/10.1002/hfm.4530030405
[38] ABB Robotics, "Multiple Vulnerabilities in ABB RobotWare," 2016. [Online]. Available: https://library.e.abb.com/public/a6b4cd9bf68c4f2f917365d3b4e32275/SI20107\%20-\%20Advisory\%20for\%20Multiple\%20Vulnerabilities\%20in\%20ABB\%20RobotWare.pdf

## APPENDIX A
## SURVEY ANSWERS

As far as we know, the only user survey about the safety or the security of industrial robots in literature was carried out in 1993 and was mainly focused on safety [37]. As summarized in Section II, we contacted 50 experts in the field, choosing among:

- researchers in the field of robotics,
- engineers working with top robot manufacturers (e.g., ABB, KUKA, Comau),
- experts from companies whose core business is based on using industrial robots,
- two IEEE RAS TCs ("Energy, environment, and safety issues in robotics and automation," and "Safety, security and rescue robotics"),
- the chair of the "Industrial Robotics" Topic Group at EU Robotics,
- two prominent mailing lists.

We received answers from 20 subjects. Some of them followed up via email for a more in-depth conversation.

Given the nature of our questions, we required the users of the survey to express a strong opinion on the subject, in order to avoid a possible central tendency bias.

An interesting result of the survey is that the overall care about safety, and the use of safety measures since the time of publishing of the last survey on safety [37], has increased, with $60\%$ of the respondents employing some customized form of protection measures (i.e., human tracking by sensors, electronic defenses). The same cannot be said for cyber security: assessments on the ICT infrastructure have been conducted only by $\sim 23\%$ of the respondents, and the share of respondents that conducted security assessment of the networks that controls the robot operations is just the $\sim 11\%$.

Moreover only $47\%$ of the respondents consider a cyber attack against industrial robots a realistic threat: of these, only 1 respondent is a developer working in the industry and 1 has experience with robots both in both an industrial and academic context.

## APPENDIX B
## COORDINATED DISCLOSURE

We reached out to the vendor in order to promptly disclose any vulnerabilities found during the research described in this article, and offered a possibility to share their remarks in the final version of this article. Following our disclosure, the vendor (ABB) confirmed and patched (or mitigated) all the vulnerabilities described in this paper. Also, the vendor released an official security advisory [38] to give the users time to update the installed base before the publication date.

We also disclosed the vulnerability in the industrial router to eWON, who fixed the issue in the latest firmware revision (11.2s2).

Table IV
SURVEY RESPONSES: GENERAL QUESTIONS

| | | |
|---|---|---|
| **Context of experience with Industrial Robots** | | |
| | Academia | 11 |
| | Industry | 6 |
| | Both | 2 |
| **User's role** | | |
| | Researcher | 8 |
| | Developer | 5 |
| | Professor | 4 |
| | Student | 1 |
| | Other | 1 |
| **Type of industrial robots used or deployed in a factory: (multiple choice)** | | |
| | Articulated | 18 |
| | Dual-arm | 6 |
| | SCARA | 4 |
| | Cartesian | 3 |
| | Cylindrical | 2 |
| | Other | 2 |
| | Delta | 1 |
| **Robots application: (multiple choice)** | | |
| | Material Handling | 13 |
| | Assembly | 13 |
| | Palletizing | 5 |
| | Other | 4 |
| | Welding | 2 |
| | Painting | 1 |
| **Industrial robot vendors employed: (multiple choice)** | | |
| | Kuka | 11 |
| | ABB | 9 |
| | Other(s) | 9 |
| | Fanuc | 6 |
| | Comau | 3 |
| | Motoman | 3 |
| | Denso | 1 |
| | Adept | 1 |
| | Kawasaki | 1 |
| **# of robots employed for development and testing** | | |
| | 1 to 5 | 12 |
| | 5 to 10 | 4 |
| | 10 or more | 2 |
| **# of robots employed in production (where applicable):** | | |
| | 10 or more | 5 |
| | 1 to 5 | 2 |
| | 5 to 10 | 1 |

Table V
SURVEY RESPONSES: SAFETY

| | | |
|---|---|---|
| **Typical safety setup:** | | |
| | Collaborative robot | 11 |
| | Caged robot | 10 |
| | Other | 3 |
| **Default safety measures are too limiting for specific use case:** | | |
| | Yes | 5 |
| | No | 12 |
| **Safety measures are customized:** | | |
| | Yes | 11 |
| | No | 7 |

Table VI
SURVEY RESPONSES: ROBOT PROGRAMMING

| | | |
|---|---|---|
| **A development cycle is employed for robot programs and automation scripts:** | | |
| | Yes | 7 |
| | No | 4 |
| **Access control policy is enforced on the robot:** | | |
| | Yes | 13 |
| | No | 5 |
| **Employees are accountable for changes to the robot's program code:** | | |
| | Yes | 9 |
| | No | 9 |

Table VII
SURVEY RESPONSES: PERCEPTION OF THE RISK

| | | |
|---|---|---|
| **Robots are connected to an internal network:** | | |
| | Yes | 14 |
| | No | 4 |
| **Robot controllers are accessible via Internet:** | | |
| | Yes | 5 |
| | No | 13 |
| **Robot controllers are accessible via a wireless network:** | | |
| | Yes | 8 |
| | No | 10 |
| **Robot controllers are connected to the ICT infrastructure via:** | | |
| | EtherCAT | 1 |
| | Intranet over Ethernet | 3 |
| | WiFi | 1 |
| | Bluetooth for discovery of non-wifi components | 1 |
| | Disconnected | 3 |
| | Internal factory VPN | |
| **The ICT infrastructure has been ever audited:** | | |
| | Yes | 4 |
| | No | 13 |
| **The network that controls robots operations has been ever audited:** | | |
| | Yes | 2 |
| | No | 15 |
| **User considers cyber attacks against robots a realistic threat** | | |
| | Yes | 8 |
| | No | 9 |
| **If a realistic threat, the consequences could be:** | | |
| | Impact on physical safety | 7 |
| | Production losses | 4 |
| | Small defects introduced in the final product | 1 |
| | Other | 3 |
| **Worst case scenario in case of insider threat:** | | |
| | Potential safety hazards, harm to human operators | 6 |
| | Mechanical damage | 6 |
| | Stop productivity | 2 |
| | IP violation/leak sensitive know-how | 1 |
| **Financial impact of an attack against robots:** | | |
| | Quantifiable | 3 |
| | Hardly quantifiable | 1 |
| | Not quantifiable | 6 |
| **If not robots, the most valuable asset at risk could be:** | | |
| | Intellectual property | 5 |
| | Humans | 2 |
| | Material goods and equipment | 2 |
| | Production data | 1 |
| | Other sensitive data (i.e., patient data for medical robots) | 1 |

# APPENDIX C
## PROGRAM TASK SOURCE CODE EXAMPLE

```
MODULE myStopRoutine
  CONST jointtarget p0 := [ [ -179, 90, -90, 0, 0, 0], [ 0, 9E9, 9E9, 9E9, 9E9, 9E9] ];
  CONST jointtarget p1 := [ [  179, 90, -90, 0, 0, 0], [ 0, 9E9, 9E9, 9E9, 9E9, 9E9] ];
  VAR jointtarget jInitial;
  VAR jointtarget jFinal;
  VAR intnum pers1int;
  VAR intnum monit1int;
  PERS bool trapped;
  PERS bool done;

  PROC main()
    TPErase;
    trapped := FALSE;
    done := FALSE;
    MoveAbsJ p0, v2000, fine, tool0;  ! move robot to initial position
    WaitRob \ZeroSpeed;               ! wait for robot to finish movement
    CONNECT pers1int WITH stopping;   ! add interrupt handler
    IPers trapped, pers1int;          ! set interrupt to trigger via watchdog on persisten variable
    CONNECT monit1int WITH monitor;   ! another interrupt
    ITimer 0.1, monit1int;            ! set interrupt to trigger after 1ms
    WaitTime 1.0;
    MoveAbsJ p1, vmax, fine, tool0;   ! move from p0 to p1 at max speed
  ENDPROC

  TRAP stopping
    VAR num delta;
    ISleep pers1int;
    jInitial := CJointT();            ! read joints coordinates
    StopMove \Quick;                  ! stop robot
    WaitRob \ZeroSpeed;
    jFinal := CJointT();              ! read joints coordinates
    delta := jFinal.robax.rax_1-jInitial.robax.rax_1;
    TPWrite "Delta = " \Num:=delta;
    TPWrite "Max acc = " \Num:=200*200/delta/2;
    IWatch pers1int;
  ENDTRAP

  TRAP monitor
    ISleep monit1int;
    jInitial := CJointT();
    IF done = false AND jInitial.robax.rax_1 > 0 THEN           ! set interrupt to trigger when the first joint reaches
or goes over halfway
      trapped := TRUE;
      done := TRUE;
      TPWrite "Checking... " \Num:=jInitial.robax.rax_1;
    ENDIF
    IWatch monit1int;
  ENDTRAP
ENDMODULE
```