Journal of Computer Virology and Hacking Techniques (2019) 15:161–175
https://doi.org/10.1007/s11416-019-0329-2

ORIGINAL PAPER

Security of controlled manufacturing systems in the connected factory: the case of industrial robots

Marcello Pogliani[1] · Davide Quarta[1,2] · Mario Polino[1] · Martino Vittone[1] · Federico Maggi[3] · Stefano Zanero[1]

Received: 18 September 2018 / Accepted: 4 January 2019 / Published online: 13 February 2019

Abstract

In modern factories, "controlled" manufacturing systems, such as industrial robots, CNC machines, or 3D printers, are often connected in a control network, together with a plethora of heterogeneous control devices. Despite the obvious advantages in terms of production and ease of maintenance, this trend raises non-trivial cybersecurity concerns. Often, the devices employed are not designed for an interconnected world, but cannot be promptly replaced: In fact, they have essentially become legacy systems, embodying design patterns where components and networks are accounted as trusted elements. In this paper, we take a holistic view of the security issues (and challenges) that arise in designing and securely deploying controlled manufacturing systems, using industrial robots as a case study—indeed, robots are the most representative instance of a complex automatically controlled industrial device. Following up to our previous experimental analysis, we take a broad look at the deployment of industrial robots in a typical factory network and at the security challenges that arise from the interaction between operators and machines; then, we propose actionable points to secure industrial cyber-physical systems, and we discuss the limitations of the current standards in industrial robotics to account for active attackers.

Keywords Industrial robots · Cyberphysical systems · Industry 4.0 · Cybersecurity · Industrial internet of things

1 Introduction

The manufacturing industry is nowadays heavily automated and integrated with business processes: The pervasive interconnection of IT systems is paving its way toward the factory, where once-isolated operational technology (OT) systems are now tightly integrated among themselves and with IT systems [5]. Devices such as industrial robots and programmable logic controllers are at the heart of the industrial internet of things (IIoT), where once air-gapped devices are now widely interconnected with factory IT systems and, ultimately, with the Internet. With good reason, in the OT ecosystem, the primary concern is the safety of the environment and the operators, as well as the integrity of the manufactured product—even when faults, human errors, or other abnormal conditions occur. However, the behavior of an active and smart adversary is different from the effect of a fault: Focusing on safety does not necessarily improve the security of IIoT systems, where, on the other hand, a security breach may easily lead to a safety issue.

In the last decade, high-profile events highlighted the importance of addressing security concerns in different classes of safety-critical, cyber-physical systems such as in the automotive [11,20] and medical industries [6,29]. In the fields of industrial control systems and critical infrastructure, the most famous and studied targeted attack against a industrial control system (ICS) was Stuxnet [8], followed by other high-profile incidents: To name a few, in 2014, an attack to a German steel mill caused the inability to shut down a blast furnace; in December 2015, a cyberattack was

✉ Marcello Pogliani
marcello.pogliani@polimi.it

Davide Quarta
davide.quarta@eurecom.fr

Mario Polino
mario.polino@polimi.it

Martino Vittone
martino.vittone@mail.polimi.it

Federico Maggi
federico_maggi@trendmicro.com

Stefano Zanero
stefano.zanero@polimi.it

1 Politecnico di Milano, Milan, Italy

2 EURECOM, Biot, France

3 Trend Micro Inc., Milan, Italy

allegedly responsible for power outages in Ukraine; more recently, between August and December 2017, researchers found in the wild instances of TRITON/TRISIS, an advanced malware with an OT payload specifically targeted against a safety instrumented system deployed in a Middle Eastern critical infrastructure [25].

Industrial robots are a key, multi-purpose cyber-physical system used in the manufacturing industry for various applications ranging from welding, pick-and-place tasks, painting, to assembly. Industrial robots are extremely widespread in industries of all sizes. Indeed, the International Federation of Robotics forecasts that more than 3 million robots will be deployed in factories all over the world by 2020, with a 14% yearly growth rate [19]. Industrial robots are complex controlled devices, programmable in a flexible way, and are evolving fast. They are now interconnected (e.g., for monitoring and programming purposes), and the innovation trend is moving them “closer” to humans. Indeed, while once robots were physically separated from human workers with a metal fence for safety reasons, smaller “collaborative” robots (or “co-bots”) are now gaining traction. Co-bots are designed to operate close to human workers, without any physical separation of their respective working spaces.

This evolution is, on the one hand, increasing the robot’s cyber-attack surface, and on the other hand, worsening the consequences of an attack. In 2017, academic [26] and industrial research [10,21] showed how even simple software vulnerabilities in networked industrial robots create the avenue for robot-specific attacks, with their impact ranging from “stolen intellectual property” to “interrupted production,” eventually creating a long-lasting negative impact on the quality and availability of the manufactured goods or on the safety.

In this paper, we take in consideration this evolutionary trend and propose a comprehensive analysis of the attack surface of modern industrial robots, as a representative example of a control system, in the context of a modern factory. Particularly, we extend our previous experimental analysis [26] with attacks and vulnerabilities originating from the human-to-machine interaction, considering the attack surface exposed by the human–machine interface (HMI) and the programming capabilities of the machine. In summary, this paper proposes the following contributions:

- We analyze the attack surface of modern industrial controllers used for robots. We focus either on the networked attack surface and on the physical attack surface that is “digitally” exploitable through the interaction with the operator;
- We analyze the role of domain-specific programming languages in the security of industrial controllers, and discuss how some “powerful” language features increase the attack surface;
- To ground our analysis, we present case studies on the control software by ABB and Universal Robots, and we use such case studies to describe concrete attack vectors.

Note. This paper is an extension of a previous conference paper [26], where we performed a security analysis of an industrial robot, mainly looking at the exploitation of the network attack surface through vulnerabilities in the robot controller’s firmware. Here, we summarize and extend the results by focusing on the broader attack surface not considered in our previous work, which includes *(a)* the physical attack surface exploitable by digital means through the user interaction; *(b)* security implications of the robot’s programming languages; and *(c)* a generalization of our results with a second case study on a controller by Universal Robots.

2 Industrial robots

An industrial robot, as defined by the ISO 8373 standard, is an electro-mechanical system composed by a multi-axis manipulator, a control system, a “operator interface,” and its hardware and software communication interface. The *robot controller* implements the core control functionality, ranging from path planning, to the execution of the manufacturing program, to the implementation of the control loops, as well as executing the basic safety logic. While the manipulator only contains actuators and sensors, the controller comprises one or more computer-based units that run a blend of general-purpose and real-time operating systems: For example, KUKA robots embed various versions of the Microsoft Windows operating system, coupled with a VxWorks-based co-processor; ABB robots use VxWorks; Universal Robots co-bots are instead based on Linux. Besides this, the controller usually includes units responsible for power supply, electrical drive, and hardwired safety logic. The operator interface, called *teach pendant*, is used by trained workers to manually control the robot or reprogram it for new tasks. The connection between the teach pendant and the controller may be either via cable, or wireless, such as in some implementations of the COMAU WiTP [9].

Traditionally, robots are “caged”: They are physically separated with a fence from the humans’ working space. Recently, there is an increasing demand for collaborative robots, where this separation does not exist, or is implemented only virtually. In collaborative robotics, due to the close proximity to humans, safety is the most important concern. Due to the lack of a cage, safety is accomplished by a mechanical design aimed at avoiding injuries, as well as speed and power limits compared to traditional robots, coupled with software-based controls (e.g., collision detection). In the collaborative robotics context, software has an even more important role in managing and guaranteeing operational safety for human-robot collaboration [17,33].

2.1 The industrial robot ecosystem

Far from being stand-alone and "air-gapped" devices, modern industrial robots are deeply integrated with the factory and embedded in an ecosystem together with other "smart" Industrial Internet-of-Things devices, which extend their capabilities and provide network interconnection. This, wider, ecosystem, exposes a large attack surface, which increases the attack surface and the attractiveness as a target of manufacturing devices such as robots.

Networked Robots Modern industrial robots are interconnected for purposes ranging from remote programming and maintenance, as specified in the standard ISO 10218-2:2011, to integration with other factory systems; more recently, the ecosystem is expanding to the direct collection of production and manteinance data from controlled manufacturing systems to cloud-based data analytics systems. Thus, robots are equipped with Ethernet ports for connectivity with the factory's local area network, and can be equipped with specialized devices, dubbed "industrial routers" or "industrial control gateways" that—similarly to consumer embedded Internet gateways—interconnect an industrial device (e.g., a robot or PLC) with a VPN or a cellular network. Besides acting as network gateways, industrial routers usually provide programmable logging, monitoring and alerting services. Industrial routers are also used to provide remote access to the vendor as part of support contracts (e.g., ABB's "service boxes").

Programming For Robots Industrial robots run complex *task programs* written in a variety of proprietary vendor-specific languages, or Domain Specific Language (DSL), such as: RAPID by ABB, KRL by KUKA, PDL2 by COMAU, and AS by Kawasaki. Task programs are written offline (i.e., on a computer) and subsequently loaded to the robot controller, or online, using the teach pendant to interactively develop the program in a "teaching by showing" fashion. Despite being specific to the industrial robotics domain, these languages offer powerful features: They contain built-in or optional primitives to access a wide range of resources and peripherals, usually with little or no intermediation and checks performed by the runtime. Sometimes, robot controllers need to exchange information between each other or with other machines on the factory line. In fact, modern languages include network socket functionalities, either by default or through optional software packages sold separately. Sometimes, robot controllers are designed to communicate directly over a network, even without external middleware or devices. For example, COMAU's PDL2 language includes documented e-mail primitives to send notifications or execute authenticated commands [12].

Robots as a Platform Besides their ability to execute custom task programs, modern robots are a complex platform that can be extended and integrated with third-party hardware and software. Vendors like ABB and Universal Robots provide full-fledged Software Development Kits (SDK) to allow customers to build complex software applications, including custom user interfaces and applications that run on the teach pendant. Additionally, robots provide platforms that allow third-party integration of hardware and software components and add-ons. As an example, Universal Robots Plus[1] consists of a range of third-party products that integrate with the robot controller's software and hardware. Those add-ons range from classic accessories such as grippers and end-effectors, toward intelligent products such as cameras and "smart" wireless safety devices aimed at increasing the interaction between workers and collaborative robots. For instance, Alumotion's YouRing is a third-party safety system that integrates, through a Bluetooth connection, with Universal Robots co-bots, and is aimed at increasing the safety of a human operator interacting with the robot, providing immediate visual feedback into the actions of the system. Both newer "IIoT" devices and components in the standard robotic architecture such as position sensors and joints are connected with the robot ecosystem and with the external world, and equipped with their own firmware that can increase the robot's attack surface or create new avenues for attacks.

3 Risks and threats to controlled manufacturing systems

The effects of an attack involving the OT ecosystem are radically different than the goals of those against IT systems. Indeed, devices in the OT ecosystem directly interact with the physical world and interfere with it. This holds true especially when we consider attacks against controlled manufacturing systems. In this context, the most important security properties are not related to data confidentiality, integrity and availability, but rather to the availability and integrity of the physical process and safety of the environment. The effect of a cyberattack can vary from halting the production plant, to altering the production outcome (e.g., injecting faults and micro-defects in the production to cause immediate or delayed financial loss), to physical damage of the system itself, or injuries to workers.

To reason on the impact of such an attack, we start from the fact that a controlled manufacturing system is composed

[1] https://www.universal-robots.com/plus/.

by sensors, actuators, and a control system, and must satisfy, at all times, three basic requirements:

– **accuracy**, i.e., the requirement to accurately "read" precise values from the physical world through sensors, and "write" correct and accurate actuator commands);
– ensuring at all times **safety** for the operators working with the robot;
– maintaining **integrity** of the machine, i.e., the controlled manufacturing system should not perform actions with self-damaging outcomes.

We consider an attack any violation of these requirements, if initiated through a digital vector.

3.1 Threat scenarios

In this section, we outline four main representative threat scenarios, which can be fulfilled through attacks against accuracy, integrity and safety requirements.

Production Outcome Altering (accuracy): An attacker may want to inject faults and microdefects in the production to cause immediate or delayed financial loss, or damage the company reputation, resulting in an advantage for competitors. Depending on the manufactured goods, defects can also cause fatalities (e.g., in automotive, transportation, or military fields). For instance, researchers showed that, by attacking the 3D printing process during the manufacturing of drone components [4], it is possible to introduce a structural and non-visible micro-defect in a drone's propeller. Indeed, they were able to reduce the component's robustness, causing an early wear out of the propeller, with catastrophic consequences on the drone's flight. Similar attack consequences can be devised for other manufacturing systems, such as CNC machines and robots.

A slight variation of this threat scenario involves the concept of *cyber-physical ransomware*: According to the thoroughness (e.g., cost) and nature of quality control implemented in the manufacturing process, attackers can create a ransom scheme that locks or degrades the production, or creates safety hazards, until a ransom is paid, similarly to what has been envisioned for other classes of industrial control systems, such as PLCs [16]. Besides this, more sophisticated schemes tailored to the manufacturing industry are possible: For instance, an attacker can sabotage part of the manufactured goods by injecting small defects, while keeping track of the sabotaged instances. Once the ransom is paid, the malicious actor will disclose a way to recognize the damaged goods, saving the company from recalling the whole production batch—or decreasing their reputation.

Production Plant Halting (accuracy, safety and/or integrity): After a cyber-physical attack, according to the extent of the damages and to the time to repair, the production may be promptly restarted or not. The downtime costs are difficult to estimate, and vary greatly according to the type and size of the targeted company. Indeed, the vice president of product development at FANUC once stated that "unplanned downtime can cost as much as $20,000 potential profit loss per minute, and $2 million for a single incident" [13].

Physical Damage (integrity and/or safety): An attacker may damage machinery (the robot itself or other factory equipment), or, worse, cause injuries to people working in the factory—for instance, by disabling or substantially altering safety devices. Causing safety hazards is by far the most impactful scenario for a cyber-physical system, with consequences far worse than production losses—and possibly requiring a long production halt while the cause of the safety hazard is being investigated and solved. For example, industrial robots must adhere to strict safety standards[2] and implement safeguards that limit the safety impact of cyber-attacks, even though they were not originally thought as countermeasures for active attacks. Nonetheless, and non-standard-compliant deployments aside, in the case of collaborative robots the physical separation from humans simply does not exist. Although co-bots are designed to be intrinsically safe even without the cage (and are actually safe in their normal operating conditions), they are not necessarily small: For instance, FANUC CR-35iA has a payload capacity of up to 35 kg. Indeed, it has been shown [7] that some collaborative robots produce enough torque to hurt a human, and researchers [3] demonstrated an attack against an Universal Robots co-bot that remotely overrode safety parameters.

Unauthorized Access (data confidentiality): Even though a manufacturing system is a cyber-physical system, its controller contains sensitive data, such as the source code of the control programs (e.g. robot's task programs, or G-code for additive manufacturing and CNC systems), which can be reverse engineered to reveal industrial secrets, and information about production schedules and volumes. Thus, the usual data security threat scenarios (e.g., unauthorized access to confidential data) apply: An attacker can simply steal sensitive data from the controller's storage, without deep cyber-physical consequences, yet impacting the victim's business.

3.2 Controller subsystems and vectors

Extending on one of the main contributions of our previous work [26] (a set of "templates" of industrial-robot-specific attacks), in this paper we observe that industrial robots are just an instance of a controlled manufacturing device,

[2] ISO 10218-1:2001 and ISO 13849-1:2008 for "caged" robots, and ISO/TS 15066:2016 for collaborative ones.

Table 1 Elements to consider when performing risk and security analysis of an industrial control system, with concrete examples drawn from the industrial robot case

Subsystem	Attack example	Attack vectors
High-level Control	Manipulating the production logic executed by the machine to introduce flaws into the workpiece, by changing the executed code or exploiting the DSL primitives	Compromise of a DSL program or library; Substitution of the control program
Low-level control	Altering the parameters of the PID control system, so the controlled actuators moves unexpectedly or inaccurately	Configuration file tampering
Calibration	Changing the calibration information to make the controlled actuators move unexpectedly or inaccurately	Calibration configuration tampering
Controller Status	Manipulating the status of the controller, for example by placing the control system in automatic mode rather than manual mode, so the operators loses control, or can get injured	HMI or controller compromise
User Interface	Manipulating the status information displayed to the user, so the operator is not aware of the true status of the controlled system (e.g., the operator thinks that the motors are off when instead they are turned on)	HMI compromise

characterized by sensors, actuators and a control system. Abstracting the set of robot-specific attacks in the more general context of a control software for smart manufacturing, Table 1 summarizes the main subsystems of a manufacturing system, alongside with examples of attacks and attack vectors.

4 Attack surface analysis

In this section, we analyze the attack surface of a modern controlled manufacturing system, and we present, with the help of case studies from industrial robots, how it can be leveraged to gain the level of access required to compromise the assets summarized in Table 1 and carry out cyber-physical threats.

We consider a knowledgeable, resourceful attacker interested in carrying out attacks targeted to a specific manufacturing system. We consider that the attacker has unlimited access to the control system's documentation and software, and has at least a partial knowledge of the processes implemented by their target. These conditions are reasonable, given that sometimes such documentation is easy to access if not publicly available online, and that prominent attacks against industrial control systems, such as TRITON, already demonstrated that sophisticated attackers are able to obtain or reverse-engineer proprietary systems, process and protocols, or to leverage insider knowledge to develop advanced malware for critical industrial systems.

To analyze the attack surface, we refer to a generic and vendor-agnostic schema of controlled manufacturing system (Figure 1), where we emphasize the input and output interfaces and protocols, as well as the interaction with human users and other cyber-physical systems. The actuation subsystem controls the physical process directly or by means of devices such as robot end effectors and tools, and is connected to its controller that provides the electrical drive. The controller is equipped with an operator interface or HMI (e.g., a teach pendant in the case of industrial robots), and optionally manages the devices connected to the actuation subsystem through dedicated proprietary connections. The controller, together with the HMI, manages the execution of the control program, and provides status LEDs and safety features such as electrical stops and the switch between automatic (high speed) and manual-controlled (lower speed) modes. The manufacturing system is integrated into a factory ecosystem and "connected" (for maintenance, programming, and to centrally control and coordinate the production) via multiple hardware communication interfaces and multiple communication protocols that allow a controller to exchange data with other controllers, factory systems, workstations, and remote

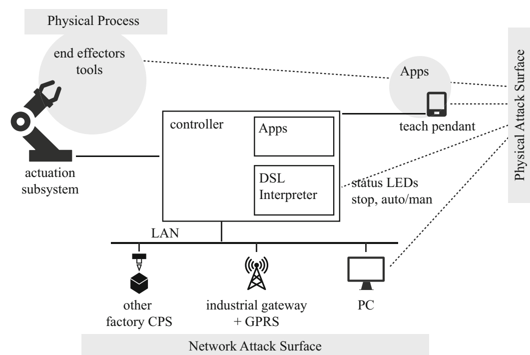

Fig. 1 Attack surface of a generic controlled manufacturing system. We emphasise the communication interfaces and interactions with the human operator and other controllers and factory cyber-physical systems

systems. Here, we consider two main classes of attack surface:

– a **network** attack surface, where a malicious actor accesses the control system, directly or indirectly, from the corporate network or from the Internet;
– a **physical** attack surface, where a operator physically interacts with the control system to carry out unauthorized operations, either via the HMI or by physically connecting devices (e.g., a USB device) to the control system. In particular, for physical access, the attacker may be either an "insider" (i.e., a factory worker with limited access to the controller) or an unauthorized person who breaks the physical security of the factory.

However, even in case of physical access, our scope includes only attacks implemented via digital vectors, or by interacting with the robot's external interface. Our attacker model does not include tampering with the physical security of the control system (i.e., picking the lock of the controller case).

4.1 The network attack surface

In a modern factory, industrial manufacturing systems are *connected* primarily for remote programming and maintenance purposes, as mentioned by the ISO standards (ISO 10218-1 and ISO 10218-2), and ultimately for centralizing control and coordinating the production; thus, they expose a large network attack surface, both to the local area network and, sometimes, remotely.

Local Area Network Controllers of industrial machines are connected in a factory TCP/IP network, for programming purpose (e.g., via a workstation connected to the factory LAN), as well as integration with other factory and information systems.

In more modern architectures, the integration with external systems is realized by means of Application Programming Interface (APIs) and middleware. In fact, modern industrial machines are equipped with rich HMI and complex APIs, essential to integrate them with the factory IT ecosystem. For instance, controllers of ABB robots expose the Robot Web Service APIs [2], a set of HTTP REST interfaces to integrate third-party programs with the controller; products by Universal Robots expose several services accessible over TCP/IP that allow to execute various actions: For instance, the "Dashboard Server" supports various commands, such as the ones to load, start and stop programs, shut down the controller, and query the robot status; the "Remote Control" interface receives URScript commands, and exposes data representing the status of the robot.

Usually, such APIs are protected by authentication and authorization mechanisms: For example, ABB implements a mechanism, dubbed User Authentication System (UAS), whereby authorized users are assigned a username and a password, and can be granted a set of permissions (grants). This system is used to authenticate access to the Web Services APIs, as well as to the proprietary protocol (RobAPI) used for the communication between the teach pendant and the controller, between the computer running RobotStudio and the controller, and to the teach pendant user interface.

The authentication and authorization mechanisms are a critical point in protecting the robot if the APIs are meant to be externally exposed, or if they are exposed behind a weakly secured LAN or a vulnerable industrial router. In general, industrial machines are directly connected to a dedicated factory network that, in the best case, is separated from non-production-critical endpoints and from the Internet, and connected only to other production-critical machinery (e.g., robots and manufacturing systems, hardened workstations for programming and control). Although directly exposing critical systems to the Internet may seem unrealistic at a first glance, during our research [26], we found a few instances of directly Internet-exposed industrial robots through searches on services that index data from Internet-wide scans (e.g., Shodan, ZoomEye, Censys). More in general, projects such as Shodan's ICS radar[3] provide insights on the non-trivial amount of industrial control systems and PLCs directly exposed to the Internet. As shown by [15], a significant amount of probing activities targeted at cyber-physical systems is happening in the wild, both for research and for nefarious purposes.

Middleware Middleware platforms simplify the development of complex multi-device applications involving multiple heterogeneous components (e.g., sensors, robots, actuators). The most prominent robotics-oriented middleware is Robot Operating System (ROS) [27], a publish-subscribe distributed middleware originally developed within the autonomous robotics community, and widely used for both research and real-world deployments.

A recent Internet-wide scan performed between December 2017 and January 2018 [14] revealed that over 100 instances of ROS master nodes are accessible from the public Internet, and that the majority (over 70%) belong to research or university networks, confirming the use of ROS mostly as a research platform. However, ROS is gaining traction also in industrial robotics, with projects such as ROS-Industrial[4] and various minor open-source tools available throughout the Internet to integrate commercial, industrial robots with ROS (e.g., open_abb[5]).

[3] https://ics-radar.shodan.io/.

[4] http://rosindustrial.org.

[5] https://github.com/robotics/open_abb.

Born as a research platform, ROS did not originally include built-in transport security, encryption or authentication, instead assuming that the network where ROS nodes are connected is completely trusted. However, guaranteeing that a network, albeit internal, is trusted, is challenging: Thus, while security was explicitly not considered in the original design of ROS, this "middleware-exposed attack surface" should be considered in future in the roadmap of ROS. Indeed, approaches to provide authentication and encryption of the communication between ROS nodes are under active development (e.g., Rosbridge, SROS, SROS2). Furthermore, ROS2 is built on top of the DDS middleware, which recently included a specification for data security [24].

Remote Networks Controlled manufacturing systems are often connected with remote networks by means of dedicated remote access devices ("industrial router", "industrial control gateway", or "service box"). Such devices enable remote monitoring and manteinance, and are often provided as part of the machine vendor's support contract for remote assistance and troubleshooting services. In this case, industrial routers connect to the vendor's network through a VPN or a cellular network, using vendor-specific or carrier-provided mobile Access Point Names (APNs). The convenience of remote management could make robots accessible through the service network, even if isolated. Indeed, the results of our Internet-wide scans show that industrial routers are easily found exposed to the Internet (more than 80,000 instances), even without authentication (more than 5,000 instances) as of late March 2017 [21]. The results of this scan are conservative, as they consider only routers that expose an easily "fingerprintable" web-based interface to the public Internet on the standard HTTP port. There may be more devices exposed indirectly or on non-standard ports. It is not surprising to find exposed industrial routers, as their purpose is to provide secure remote access to industrial devices. However, these devices offer a considerable remote attack surface: If vulnerabilities are present, they can be exploited to gain access to robots, PLCs, and other industrial control systems. Indeed, a superficial security analysis of 12 different industrial routers by means of static analysis [21] revealed that vulnerabilities and misconfigurations in industrial routers are still easily found (e.g., outdated software components with known vulnerabilities, default credentials, poor authentication and transport encryption, vulnerabilities in the web interface).

4.1.1 Case studies

Our previous conference paper [26] and various industrial research reports [3,10] thoroughly analyzed the security of modern industrial robots from a network attacker standpoint, concluding that critical software vulnerabilities in network services running on robot controllers are far from being rare, including memory corruption issues, and logical vulnerabilities introduced at design time, especially in critical functionalities such as auto-configuration, update, or the interaction with plug-ins.

ABB RobotWare As shown in [26], ABB RobotWare (version 5.x) did not check FTP credentials during the controller boot to allow the FlexPendant to download software and configuration data; they also included a fixed hard-coded password to implement auto-configuration of the service box, by allowing it to execute a set of commands once connected. Combining together these vulnerabilities, along with some non-authenticated exploitable buffer overflows, allowed to bypass the robot's user authentication, execute malicious code, and create powerful robot-specific attacks. To make things worse, various vulnerabilities found in popular industrial network gateways demonstrated the feasibility of fully remote attacks to manufacturing systems.

Universal Robots: "DSL Request Forgery" As a second case study, we present an attack on Univeral Robot cobots, caused by logical design issues in the design of the controller (e.g., lack of privilege separation) coupled with powerful capabilities of the robotics programming language. We perform the experiment described in this paragraph, and the other experiments on Universal-Robots described in the remainder of this paper, by analyzing the firmware images available for download from the support website. We used the CB 3.1 (USB stick version) 3.4.5-100 software image that we executed in a virtualized environment.

Controllers by Universal Robots interpret programs developed in a proprietary language, URScript. Among the capabilities of this language, programmers are free to use low-level network sockets or higher level remote procedure calls (XML-RPC) for network communication. Those features are widely used in both standard controller components and URCap plugins; furthermore, it is a common pattern to exchange URScript commands directly over non-authenticated network sockets.

The software architecture of a controller by Universal Robots exposes various networked services accessible over TCP/IP, including a "dashboard server" and a control program, URControl, that accept network connections. All the software running on the controller is executed by a highly privileged operating system-level user (*root*), violating the principle of least privilege. As discovered previously [3], these services allow to execute several actions without authentication (e.g., stopping and starting a program, and loading a specific program) according to various protocols. By using this feature, an attacker can cause denial of service, load an arbitrary program *already present* on the controller, or exfiltrate confidential data.

One of the protocols exposed by URControl on the TCP port 30002 listens *by design* for arbitrary URScript code and executes it. URScript programs are limited in functionality. Thus, apparently, this “code execution by design” does not allow the attacker to gain full control of the controller. This feature is used, for instance, by URCap plugins. For example, Listing 1 shows how a URCap plugin sends URScript commands to the robot via a network socket, and how the commands, in turn, uses XML-RPC functionalities to make the robot execute an action (i.e., blink LEDs on the product associated to the URCap). The capability of executing unauthenticated URScript commands that can initiate network requests allows to communicate with any service and network reachable from the controller—but not from the attacker's network—increasing the attack surface.

Listing 1 Simplified example of “secondary socket” client present in a URCap plugin

```
def go_freedrive():
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.connect((cfg.ROBOT_HOST, 30002))
    s.send('def set_freedrive():\n')
    s.send('uring = rpc_factory("xmlrpc",
"http://127.0.0.1:33000")\n')

s.send('uring.set_leds_mode(2,0,100,0,100,150,150,150,0,-1)\n')
    s.send('end\n')
    s.close()
```

Furthermore, the controller runs a default installation of Apache Felix that exposes a management console on TCP port 6666, which listens only on connections from *localhost*, i.e., the controller itself.[6] This console allows clients to install and run arbitrary OSGI bundles (i.e., Java packages). Exploiting this, a network attacker can use the arbitrary URScript execution allowed by the URControl interface to execute a script that connects to the controller from the perspective of the controller itself, i.e., to localhost, and sends commands to the Apache Felix console, which is not otherwise network-accessible. As Apache Felix is running with the privileges of the *root* user, this grants the attacker unauthenticated arbitrary and privileged code execution on the robot controller, using an attack similar to a “server-side request forgery” (SSRF), which we call DSL Request Forgery attack, as it is initiated through commands in the robot's programming language.

Apache Felix is not the only service vulnerable to this issue. We analyzed publicly available URCap plugins, and found instances that install a daemon, and an associated network service, bound to localhost, and not directly reachable from the network—unless the attacker uses the above attack.

As all of the features we employed are also employed by URCap plugins, mitigations are hard to implement, making our attack extremely powerful: Any change made to the management interfaces to fix this vulnerability will have to involve not only Universal Robots, but also third-party URCap plugin developers, as well as any customer leveraging those functionalities. Indeed, as the code execution is by design, this vulnerability was not fixed, and Universal Robots recommends as remedial actions to “only allow trusted users physical access to the robot control box and teach pendant,” not to “connect the robot to a network unless it is required by the application”, and using “a secure network with proper firewall configuration” [32], suggesting that the underlying threat model of the robot considers a trusted network and trusted-only physical access as the only line of defense.

4.2 The “interaction” attack surface

The development of user interfaces resilient to a malicious user, including the interaction between operators and the HMI, as well as external ports (e.g., USB, LAN) that can be used to gain unauthorized access to the industrial controller's software, is an aspect often overlooked or not considered altogether in the threat model for controlled manufacturing systems. The consequences of malicious physical access to a controller can be overwhelming, ranging from stopping the machine or issuing malicious commands, to subverting the software running on the controller: For instance, if the controller does not properly verify the authenticity and integrity of the firmware upon every boot and during updates, a malicious operator can substitute the firmware with a tampered version; if the user interface access control is weak, a physical attacker can run malicious commands.

Physical access can also be used as an *indirect* vector: Attackers can leverage a legitimate user's physical access by luring them to carry out malicious operations via social engineering or technical means. For instance, infected USB devices can be a threat used to compromise even network-disconnected controllers: Research has shown that the handling of untrusted USB devices by users is particularly dangerous [30], and, as an anecdote, during our research we found that one of the robot controllers we had access to (a 2011 controller by KUKA based on Microsoft Windows XP Embedded, and disconnected from any network) was inadvertently infected by a USB-spreading malware. The robot was infected even though it was equipped with an antivirus software, a precaution suggested in the product documentation: antiviruses may be difficult to keep up-to-date, especially if the robot is not connected to any network, and they may be easily bypassed by unknown or targeted malware. The malware sample we found[7] was a typical and non-targeted information stealer spyware for Microsoft Win-

[6] https://felix.apache.org/documentation/subprojects/apache-felix-remote-shell.html.

[7] SHA256 hash: `78d9b449e64b4b2bb40ad30b2033420599b5923 af5ae1c00b7eb5f4447acc772`.

dows, aimed at harvesting password for online games; despite it was a generic malware rather than a robot-specific targeted threat, this incident shows the effectiveness of the USB attack vector to target industrial robots.

Furthermore, the extensibility of the smart manufacturing ecosystem allows to envision a future in which re-purposed and second-hand end effectors can become threat vectors using techniques such as BadUSB [23]: Indeed, some end-effectors and products such as Alumotion's YouRing[8] already require the user to connect a USB device to the controller (e.g., to communicate via a serial or Bluetooth adapter).

4.2.1 Case studies

In this section, we consider the digital attack vectors used by a physical attacker. In this scenario, often overlooked by the robot vendors' threat models, there are multiple attack vectors: Every possible user interaction could be used to compromise the controller, be it through the teach pendant's interface, or a custom USB device, as highlighted in the following case studies.

Weak Interface Hardening Sometimes, weak applicative access control or user interface hardening allows users to bypass the intended functionality of the interface, often resulting in full access to the underlying operating system. For example, Universal Robot's PolyScope interface (version 3.4.4.412) features a calibration interface, which can be accessed from the login screen using a pre-defined password, mentioned in the service manual [31]—meant as a weak safeguard for unintentional tampering by unauthorized operators, rather than for securing the interface against a determined attacker. We also found other hardcoded credentials not mentioned in publicly available manuals: Notably, one of these credentials enables a *Low Level Controller* that, among the various functionalities, allows to implement a custom control logic; another one allows to reset the passwords set by the user. These passwords are hardcoded, are the same for all the robots, and cannot be changed or disabled. Using the above passwords, users can access a text editor running with root privileges, which allows to edit arbitrary text files with only limited sanitization (e.g. some characters like a dot cannot be used, supposedly to prevent path traversal). This will not stop a malicious operator from accessing important files like `/etc/passwd/` and change the Unix root password hash (Figure 2), leading to complete access to the operating system.

Bad Magic In Universal Robot controllers, *magic files* are in charge of executing maintenance and management tasks,

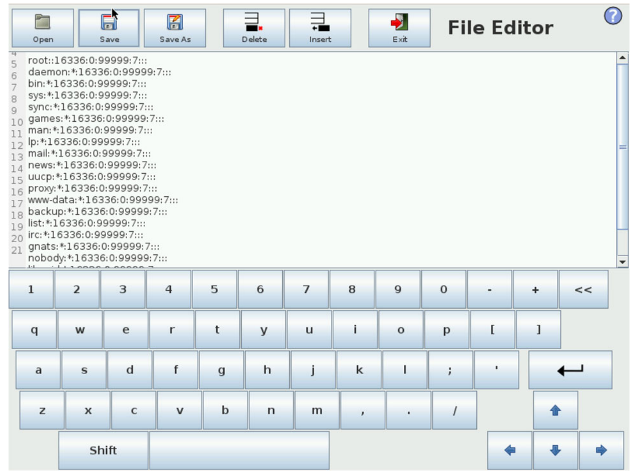

Fig. 2 The text editor running in the Universal Robot's PolyScope interface, editing arbitrary files as the root user

such as backing up log files and configurations [31]. Technically, they are simple shell scripts. The implementation of magic files allows for an automatic execution: when such a file is copied to the robot controller, it automatically executes as the root user with no integrity or validity check, just like the old and deprecated "autorun" functionality of Microsoft Windows. As a side effect, getting full code execution on a controller with this functionality is, currently, only a matter of plugging in a malicious or infected USB mass storage device.

5 Impact of domain-specific programming languages

The defining feature of controlled manufacturing systems is their ability to be quickly reconfigured for different applications, i.e., their ability to run some kind of "program", written in a specific language. Although different from general-purpose programming languages used to develop applications for general-purpose computers, such languages are relatively powerful, as they expose a rich set of APIs to ease common automation tasks.

In particular, among the various manufacturing systems, industrial robots sport one of the most complex and flexible programming environment. There, "task programs" are usually written in proprietary domain-specific languages (DSL) and interpreted by the robot controller. Task programs can be written either offline—i.e., developed in a computer-based environment, tested in a simulator and uploaded to the robot at a second stage—or online, i.e., developed interactively with the use of the teach pendant according to a paradigm known as "teach programming" (ISO 8373). Besides the basic functionalities needed to move the robot's joints, many

[8] http://tools.alumotion.eu/it/youring/.

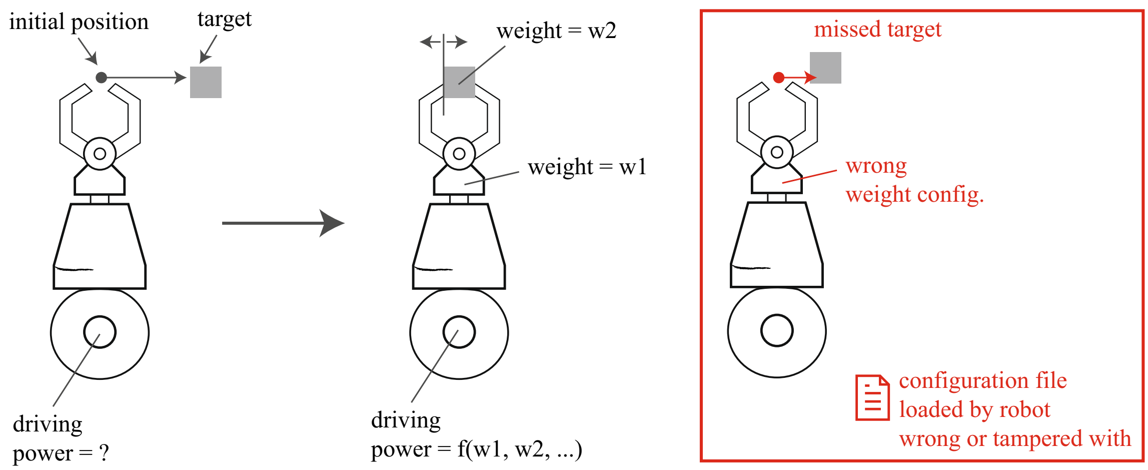

Fig. 3 "Implicit parameters" and their effect in the execution of robotics task programs

language allow to perform system-related tasks, such as acquiring input from (and displaying output to) the teach pendant, reading and writing files, and opening network sockets, either with features available out-of-the-box or by means of plug-ins sold separately.

Task Programs and Implicit Parameters The execution of a task program requires some *implicit* parameters that influence the program's execution behavior: in fact, the program's source code does not contain all the parameters and information needed for a deterministic execution, as the interpreter needs to take into consideration the parameters of the specific robot to correctly perform tasks such as path planning, and to move the robot correctly. For instance, as shown in Figure 3, to correctly compute the driving force for the mechanical arm, the controller must know the weight of the arm itself; otherwise the robot will miss the target (in case of a pick and place application). This makes the interpretation and execution of a task program more complex than the one of a program written in a common programming language.

"Appification" of Industrial Programs In the field of consumer robotics, "app stores" for robots, such as the Robot App Store[9] opened in 2011, are becoming increasingly available. This "appification" of robotics, fueled by the availability of easy-to-use APIs, is moving to industrial robotics. For example, ABB hosts the RobotApps forum[10], where users can exchange 3D models, videos, add-ins and applications; Universal Robots offers *Universal Robots+*, a vendor-vetted "online showroom" of third-party hardware and software products that can be integrated with UR robots.[11]

[9] http://www.robotappstore.com/.

[10] https://robotapps.robotstudio.com.

[11] https://www.universal-robots.com/plus.

Security Challenges This context raises two important security challenges: First, the security of the interpreter and of the environment where task programs are executed; and second, the security of task programs written in robotics DSLs. Both issues contribute to the attack surface of the robot, for what concerns the "network" attack surface, and for what concerns the "physical interaction" attack surface.

5.1 Vulnerable task programs

Task programs are, ultimately, software products: As such, they can (and do) contain software vulnerabilities. Modern robotic DSLs are equipped with features to programmatically access sensitive system resources, such as configuration files, and to issue cyber-physical commands to control the robot's movement. Furthermore, they can process untrusted input both from the robot's network- and physical- attack surface. These features result in taint-style vulnerabilities, when data from untrusted sources, such as network sockets or user input from the teach pendant interface, is sent without sanitization or security checks to sensitive sinks, such as file system operations or robot movement commands. This is worsened by the fact that task programs are written by automation and robotics experts, rather than software development teams with information security awareness and knowledge.

Networking The ability to use network sockets is an integrated or optional feature in most robotics DSLs. Task programs can be designed to be parameterized at runtime: They take input data from a source external to the controller and act according to this data. Such input may either come from the network, or from operators physically interacting with the robot by means of the teach pendant.

To implement network functionalities, most programming languages provide only basic and low-level socket-like primitives, that may be difficult to securely use, especially by robotics experts who lack specific knowledge in security and in secure software development, and easily lead, for instance, to unauthenticated and encrypted communication. This can result in safety- and production- critical information (e.g., joint positions) being exchanged over an unsecured network channel that can be possibly tampered with by an attacker with local network access. Moreover, the use of data from network sockets without sanitization easily leads to taint-style vulnerabilities.

Runtime Symbol Resolution Modern robots call for dynamic production: For example, it is possible that, when an external computer connected with a vision system detects a shape for the next part to be processed, it chooses the correct procedure to execute. This is common when the robot processes objects that are heterogeneous in shape or weight. This is technically enabled by calling dynamically procedures in the robotics DSL according to network- or operator-provided data, and many robotics languages provide support for this feature. For example, ABB's RAPID includes the functionality known as "late binding", and COMAU'S PDL2 include the keyword `CALLS`.

Listing 2 provides a simplified example of an unsafe use of network sockets and of the late binding functionality in RAPID. In the example, an external source sends the name of the function to call through a network socket (or, alternatively, through input acquired from the teach pendant). Without proper sanitization, an attacker may connect to the task program and call routines other than the intended ones. Not only the routines defined in the program by the programmer can be called, but also a small set of "system" functions can be called providing the attacker an interesting vector to carry out other attacks (i.e., performing a denial of service by executing repeatedly a `WarmRestart`, or disrupting production by using `ClearPath`).

Furthermore, ABB's RAPID also provides ways to perform dynamic code loading of an entire module from a remote resource. This is a useful feature with robots working in parallel and reproducing the same production steps, but presents security challenges.

Listing 2 Example of use of unsafe functionalities in RAPID: sockets and late binding. The code will receive a string through socket and will call a predefined routine for retrieving the current robot status.

```
PROC getCommand()
  ReceivedData := stEmpty;
  !Receive the data
  SocketReceive clientSocket\Str:=
      ReceivedData\Time:=WAIT_MAX;
  %ReceivedData%;
  ERROR
    IF ERRNO = ERR_SOCK_CLOSED THEN
      TPWrite "reconnecting...";
      ConnectionRoutine;
      sockStatus := SocketGetStatus(
                          clientSocket );
      WHILE (
          sockStatus <> SOCKET_CONNECTED)
      DO
          ConnectionRoutine;
          sockStatus := SocketGetStatus(
                          clientSocket);
      ENDWHILE
      RETRY;
    ENDIF
ENDPROC
```

Example: Vulnerable RAPID App During an analysis of the ABB application store for taint-style vulnerabilities in RAPID applications, we found a vulnerable webserver designed for ABB controllers. The software is intended to share files from the robot controller with a client; it contains functionalities for sending a whole directory, an for sending a single file. The methods access the file system. The unsanitized content of the filename in the user's request is passed through file system operations, allowing the user to retrieve arbitrary files in the whole file system rather than just the files supposed to be served (i.e., path traversal). We reported the vulnerable application to ABB product security, resulting in the application being removed from the store. At the moment of our report, it had been downloaded a few hundreds of times.

5.2 Interpreter issues

The second challenge in securing robotics programming languages is the complexity of the interpreter. Indeed, the complexity of the interpreter code (from parsing routines, to the interpretation of the task program) leads to classic software security challenges (e.g., memory corruption vulnerabilities in the interpreter code) and in more subtle issues, especially when different trust domains are involved, or when task programs are executed from untrusted sources.

Case Study: Permission Circumvention in ABB Robots Writing and editing task programs is an essential part of the interaction between the operator (or the programmer) and the robot: Operators routinely interact with, and edit task programs through the teach pendant, making them accessible from the physical attack surface. Robot controllers are meant to be multi-user, and thus equipped with an authorization system with fine-grained permissions. As in other scenarios (e.g., smartphones), the semantic differences between different, unrelated permissions, and their interaction may hide security issues.

One of such issues is the "permission circumvention", i.e., the fact that attackers who are granted only a limited set of permissions could try to "circumvent" the permission system to perform unintended or unwanted actions.

ABB controllers are equipped with a User Authorization System (UAS), based on a role-based access control system: Each user, identified by a username, belongs to a group, and each group is assigned a set of permissions. Access is granted upon password-based authentication. Instead, task programs are not bound by UAS permissions and always run with full privileges, due to the common requirement of running programs with a different privilege level than the operator who is logged into the robot controller, or of running task programs when no user is logged in at all. Thus, although the UAS restricts how the user interacts with the controller through the accessible APIs, UAS grants have no impact on the interpreted RAPID code. This stems from the assumption that programs can be only loaded by a trusted user, and this means that, when users are assigned the UAS grant to "edit RAPID code," they can modify the task program currently loaded in the execution memory.

Although users with the permission to edit RAPID code are considered trusted from the point of view of the robots movements, they are not necessarily allowed to perform administrative actions. This scenario can be considered common in different applications where few manual changes in the code can help in adjusting the output of the control loop, when automatic feedback is not helping and there is no possibility to restart the whole production cycle, or if the input parts in the production may vary between stocks and there is no automatic way of telling so.

As RAPID contains features for I/O and for executing program modules, the user can bypass other restrictions: for instance, they can read/write to the filesystem, and load in memory other modules. First, it is reasonable to give operators access only to a *single* pre-loaded program (i.e., the one that is supposed to apply to the domain/context of the specific operator), without granting them to load code belonging to other programs. However, if the operator is granted permissions to edit the current program, the operator is able to load arbitrary modules into the current program by calling from RAPID code the procedures `Load` and `StartLoad`.

We also verified that a user who is assigned only the controller grants to edit RAPID code and to execute task programs is able to write code to read (or even change) arbitrary files to the file system, even if this user is not granted any file-system access. A malicious operator can write a program that prints the content of any file in the controller file system: in the worst case, a program to read the file `HOME:/../../INTERNAL/uas\_users.xml`, containing the weakly obfuscated list of UAS users with their password, including the administrator's credentials. Being able to read and edit this file, the malicious operator can log in to the controller with full permissions; otherwise, the user could directly manipulate arbitrary configuration files, or directly change their permissions by editing `uas\_groups.xml`.

6 Mitigations and defenses

As any complex software-based device, controlled manufacturing systems often contain design errors and bugs that lead to security issues. Indeed, designing software free from errors and vulnerabilities is currently an extremely hard task, and formal verification techniques are too costly for such a complex device. However, due to the long service-life and the legacy roots of controlled manufacturing system, a simple "patch and fix" style of managing security vulnerabilities is not sufficient, calling for secure architectures that are able to guarantee safety and resilience in the presence of software compromise. Controlled manufacturing systems are controlled by a set of embedded systems, and securing them presents challenges similar to the ones found in the embedded and IoT space; they are also cyber-physical systems used for critical processes, and they present challenges similar to those of the industrial control system space [22]. In this Section, we outline the main avenues for defending controlled manufacturing systems, mostly with a long term focus, and hinting at the main open research and engineering challenges in this field.

Secure Software Development Lifecycle The process of designing an industrial controller should take cyberattacks into account. On the one hand, this translates into adopting practices such as a secure software development lifecycle [18] during the design and implementation phases of all the software components, including steps ranging from "secure coding' standards to reduce the likelihood of simple vulnerabilities, to security-oriented code review activities, up to the management of third-party dependencies. On the other hand, this requires a deep analysis of the device's attack surfaces aiming at reducing them to the minimum that preserves the intended functionality, and in considering the presence of an active attacker in all the phases of the system design. Furthermore, the secure lifecycle should include a formalized vulnerability management process and an easy way for customers or external researchers, including non-customers, to report vulnerabilities and security incidents.

Patch Management Once a security vulnerability is found, it needs to be addressed, and the resulting security update needs to be timely deployed across existing systems, in order to reduce the window of opportunity for the attacker. In an environment as critical as a factory it is not easy to apply patches to already deployed systems. First, in most cases, applying software updates requires to interrupt the production, causing costs and delays. This opens the challenge of hot-patching operating systems and user-space applications without interruption; although this challenge has been partially addressed both in research (e.g., [28]) and in commercial systems (e.g., Linux's `kpatch`), existing solutions present drawbacks, and live patching remains not yet feasible in most scenarios. Second, the majority of industrial routers and controllers do not have procedures to *automatically* update their firmware when security vulnerabilities are discovered and patched, in order to avoid connecting them to the Internet, to address legitimate concerns on regression issues, as well as to avoid possible production interruption in case an update renders the controlled system unserviceable. Thus, the decision to update the software is up to the customers: It is influenced by their awareness of security issues, and their evaluation of the cyber-security risk.

Secure and Resilient Architectures An effective defense is the adoption of more secure architectures to increase the bar of attacks and reduce their impact. This opens various engineering challenges for rendering devices more resilient to attacks, assuming the compromise of one or more components. In their most basic form, this translates in designing the system with the principle of minimum privilege, reducing the implicit trust between hardware and software components, and compartmentalizing the system so that a single vulnerability is not enough for a complete compromise. Furthermore, it is advisable to implement software signing to guarantee integrity and provenance of firmware updates as well as the software of plugin and accessories.

Indeed, various issues we found in the case studies presented in this paper stem from design decisions that consider the environment where the control system is deployed completely trusted. For example, the DSLRF vulnerability in Universal Robot can be mitigated by disallowing unauthenticated execution of URScript code from the network, and its impact can be reduced by adopting the principle of minimum privilege throughout the software running on the controller (e.g., avoid running all the processes as the root user). However, such mitigations are challenging, and should be intended as a long-term solution: In this example, the non-authenticated transmission of commands is used throughout the Universal Robots architecture and in third-party software, thus, changing the interface to require authentication would break existing deployments.

Another example of trusting the environment is the attack surface exposed to an operator physically interacting with the control system. To mitigate the issues raised by the physical attack surface, vendors should take into account that the operator is not necessarily trusted, and implement robust access control policies in their software.

Secure Deployment Besides improvements in the design of manufacturing systems, network segmentation and secure deployments are still one of the most important avenues to reduce the attack surfaces and contain the effects of an attack. To aid with this task, we advise to distribute a "secure operation manual" with each controller, with clear guidelines to help the user configure (and harden, if necessary) the robot, tailoring the security level to their needs, possibly disabling features and services not needed for the specific deployment. This could be coupled with secure defaults in the configuration of the controller software (e.g., forcing the customer to set a password during the installation procedure instead of shipping a default one), taking into account the trade-off between the device security and the product usability. Risk management standards and activities (i.e., ISO/TS 15066) could be expanded to cover security concerns raised by this paper and other recent research.

Secure Task Programs The problems concerning the programming of the machine with modern domain-specific languages can be mitigated on two levels. As with programming traditional systems, programmers should be advised to properly check and sanitize untrusted data, e.g., wrapping the late binding mechanism inside a input validation method that checks that the called procedure is among a white-list of valid targets. Unfortunately, doing so requires that the developers of task programs know the security implications of their programs, which we believe is not (yet) the case for robot programming. Thus, as a medium-term mitigations, designers of robotics programming language should develop high-level and easy-to-use abstraction that make it more difficult to insert security vulnerabilities (e.g., robotics DSL can provide basic socket abstractions to implement communication routines, instead of providing higher-level interfaces with easy data type and format checking, and default use of secure and authenticated transmissions channels). As a long-term countermeasure, we argue that smaller and more specific languages may be used in place of today's full-fledged ones, developed when robots ran in isolation: This way, it would be possible to reduce the attack surface and the likelihood for a robot programmer to introduce a critical vulnerability.

The role of standards The industrial robotics field is heavily standardized: Many ISO standards dictate safety requirements of machinery and industrial robots. For example, ISO 10218:2011 focuses on requirements in terms of firmware performance, stop functionalities, emergency stop, teach pendant controls and speed limits. On the other hand, ISO 34849 and IEC 62061 regulate the performance of safety related control systems (SRP/CS – safety related parts of control systems), taking into account also concepts related to "design and installation". Safety standards are also being extended to co-bots: the ISO/TS 15066:2016 regulates the key safety requirements of collaborative robots, such as setting bio-mechanical limits for contact with parts of the human body. However, standards do not explicitly consider the risk of cyberattacks. Safety is concerned with all the actions and measures taken to prevent injuries and accidents in normal condition: For instance, putting robots in a cage to prevent a human being to come too close to the robot; having a stop button to instantly remove electric power; implementing force sensing devices in the co-bot robotic arms, so that an accidental collision with unintended objects result in a instant stop. For what concerns collaborative robots specifically, even though they are intrinsically safe through the use of dedicated safety sensors and electronics, it should be considered that in some instances the standard-dictated limits could be exceeded [7].

Thus, to guarantee the safety of the user and all of the other assets, it is important to start considering security when interconnecting manufacturing systems. Indeed, designing a device for safety (in presence of a fault, or of an unintentional mistake) does not provide guarantees against a motivated attacker in an adversarial context—that is, when security problems arise. Even if standards consider limits to what the user can do with safety related software (which requires that the firmware must not be easily modifiable), currently, there is not a clear and systematic approach to security. Hopefully the industry will steer toward standards and best practices in security, like it already happened in the industrial control system (ICS) world with the standard ISA/IEC 62443 (ISA99)—mostly concerned with the control network security and segmentation rather than with software security—and with the NIST special publication 800-82, as a guide to industrial systems security, and in the automotive industry, with the SAE J3061, an automotive cybersecurity guidebook published in 2016, and thought also to be a foundation for further security standard development throughout the industry.

7 Conclusions

In this paper, we analyzed the attack surface of modern controlled industrial manufacturing systems, and the security risks that arise from their interconnection, operation, and expansion with accompanied IIoT devices. We discussed the physical and network attack surface of a typical controller, and how the complexity of these systems can expose flaws at different levels. We found that the attack surface is not limited to the network attack surface, but it includes the physical interaction with the HMI operator and the use of domain-specific programming languages. Indeed, we found attacks that exploits the expressiveness of DSLs and untrusted inputs, as well as weaknesses in the implementation of permission systems (often thought to avoid easy access to some features rather than to stop a determined unauthorized user), which can result in a an attacker gaining full remote control of the controller. The design and the architecture of a controlled manufacturing system is still largely similar to what it was ten years ago—and it is probably close to what we will still see for quite some time. We believe that connecting systems can improve the cooperation and is a necessary development for the modern factory, but we also believe that, to realize this vision, systems need to be robust enough not only to unintended errors but also to malicious attempts to disrupt the system.

Vulnerability Disclosure According to industry-standard coordinated disclosure practices, we disclosed to the involved vendors or to the U.S. ICS/CERT security issue we found during this research. Specifically, ABB acknowledged the issues found while working at our previous conference paper in an advisory [1], while the ICS/CERT acknowledged the issues found on the Universal Robot controller with an advisory [32], and assigning CVE-2018-10633 and CVE-2018-10635.

Acknowledgements Politecnico di Milano received funding for this project from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement nr. 690972, and has been partially supported by CINI Cybersecurity National Laboratory within the project FilieraSicura: Securing the Supply Chain of Domestic Critical Infrastructures from Cyber Attacks (www.filierasicura.it), funded by CISCO Systems Inc. and Leonardo SpA.

References

1. ABB: Cyber Security Advisory, SI20107. https://library.e.abb.com/public/a6b4cd9bf68c4f2f917365d3b4e32275/SI20107%20-%20Advisory%20for%20Multiple%20Vulnerabilities%20in%20ABB%20RobotWare.pdf (2016)
2. ABB Robotics: Robot web services. http://developercenter.robotstudio.com/webservice/api_reference
3. Apa, L.: Exploiting industrial collaborative robots. http://blog.ioactive.com/2017/08/Exploiting-Industrial-Collaborative-Robots.html (2017)
4. Belikovetsky, S., Yampolskiy, M., Toh, J., Gatlin, J., Elovici, Y.: dr0wned—cyber-physical attack with additive manufacturing. In: 11th USENIX Workshop on Offensive Technologies (WOOT 17). USENIX Association, Vancouver, BC. https://www.usenix.org/conference/woot17/workshop-program/presentation/belikovetsky (2017)
5. Bloem, J., Van Doorn, M., Duivestein, S., Excoffier, D., Maas, R., Van Ommeren, E.: The fourth industrial revolution—things to tighten the link between it and ot. Tech. Rep., SOGETI. https://www.fr.sogeti.com/globalassets/global/downloads/reports/vint-research-3-the-fourth-industrial-revolution (2014)
6. Bonaci, T., Herron, J., Yusuf, T., Yan, J., Kohno, T., Chizeck, H.J.: To make a robot secure: an experimental analysis of cyber security threats against teleoperated surgical robots (2015). arXiv preprint arXiv:1504.04339
7. Bonev, I.: Should we fence the arms of universal robots? http://coro.etsmtl.ca/blog/?p=299 (2014)
8. Brunner, M., Hofinger, H., Krauß, C., Roblee, C., Schoo, P., Todt, S.: Infiltrating critical infrastructures with next-generation attacks. Tech. rep, Fraunhofer Institute for Secure Information Technology (SIT), Munich (2010)
9. Calcagno, R., Bonivento, A.: Wireless teach pendant for robotics technological rationale for comau witp. IFAC Proc. Vol. **39**(15), 494–497 (2006). https://doi.org/10.3182/20060906-3-IT-2910.00083. 8th IFAC Symposium on Robot Control
10. Cerrudo, C., Apa, L.: Hacking robots before skynet. https://ioactive.com/pdfs/Hacking-Robots-Before-Skynet.pdf (2017)

11. Checkoway, S., McCoy, D., Kantor, B., Anderson, D., Shacham, H., Savage, S., Koscher, K., Czeskis, A., Roesner, F., Kohno, T.: Comprehensive experimental analyses of automotive attack surfaces. In: Proceedings of the 20th USENIX Security Symposium (2011)
12. Comau Robotics: PDL2 Programming Language Manual—System Software Rel. 3.3x. Comau Robotics (2009)
13. Cruz, L.: Digitization and iot reduce production downtime. https://newsroom.cisco.com/feature-content?type=webcontent&articleId=1764957 (2016)
14. DeMarinis, N., Tellex, S., Kemerlis, V., Konidaris, G., Fonseca, R.: Scanning the internet for ros: A view of security in robotics research. arXiv preprint arXiv:1808.03322 (2018)
15. Fachkha, C., Bou-Harb, E., Keliris, A., Memon, N., Ahamad, M.: Internet-scale probing of CPS: inference, characterization and orchestration analysis. In: Proceedings of the 24th Annual Network and Distributed System Security Symposium, NDSS (2017). https://doi.org/10.14722/ndss.2017.23149
16. Formby, D., Durbha, S., Beyah, R.: Out of control: Ransomware for industrial control systems. Tech. Rep., RSA Conference. http://cap.ece.gatech.edu/plcransomware.pdf (2017)
17. Fryman, J., Matthias, B.: Safety of industrial robots: from conventional to collaborative applications. In: Proceedings of the ROBOTIK 2012; 7th German Conference on Robotics, pp. 1–5 (2012)
18. Howard, M., Lipner, S.: The Security Development Lifecycle, vol. 8. Microsoft Press, Redmond (2006)
19. International Federation of Robotics: Executive Summary: World Robotics 2017 Industrial Robots. https://ifr.org/downloads/press/Executive_Summary_WR_2017_Industrial_Robots.pdf (2017)
20. Koscher, K., Czeskis, A., Roesner, F., Patel, S., Kohno, T., Checkoway, S., McCoy, D., Kantor, B., Anderson, D., Shacham, H., et al.: Experimental security analysis of a modern automobile. In: Proceedings of the 2010 IEEE Symposium on Security and Privacy, pp. 447–462 (2010). https://doi.org/10.1109/SP.2010.34
21. Maggi, F., Quarta, D., Pogliani, M., Polino, M., Zanchettin, A.M., Zanero, S.: Rogue robots: Testing the limits of an industrial robots security. Tech. Rep., Technical report, Trend Micro, Politecnico di Milano. https://documents.trendmicro.com/assets/wp/wp-industrial-robot-security.pdf (2017)
22. McLaughlin, S., Konstantinou, C., Wang, X., Davi, L., Sadeghi, A.R., Maniatakos, M., Karri, R.: The cybersecurity landscape in industrial control systems. Proc. IEEE **104**(5), 1039–1057 (2016). https://doi.org/10.1109/JPROC.2015.2512235
23. Nohl, K., Lell, J.: Badusb-On Accessories that Turn Evil. Black Hat USA (2014)
24. Object Managemenet Group: The DDS security specification version 1.1. https://www.omg.org/spec/DDS-SECURITY/1.1/ (2018)
25. Pinto, A.D., Dragoni, Y., Carcano, A.: TRITON: The first ICS cyber attack on safety instrument systems. Tech. Rep., Nozomi Networks. https://www.nozominetworks.com/downloads/US/Nozomi-Networks-TRITON-The-First-SIS-Cyberattack.pdf (2018)
26. Quarta, D., Pogliani, M., Polino, M., Maggi, F., Zanchettin, A.M., Zanero, S.: An experimental security analysis of an industrial robot controller. In: Proceedings of the 38th IEEE Symposium on Security and Privacy, pp. 268–286 (2017). https://doi.org/10.1109/SP.2017.20
27. Quigley, M., Gerkey, B., Conley, K., Faust, J., Foote, T., Leibs, J., Berger, E., Wheeler, R., Ng, A.: Ros: an open-source robot operating system. In: Proceedings of the ICRA Workshop on Open Source Software (2009)
28. Ramaswamy, A., Bratus, S., Smith, S.W., Locasto, M.E.: Katana: A hot patching framework for elf executables. In: Proceedings of the 2010 International Conference on Availability, Reliability and Security ARES, pp. 507–512. IEEE (2010). https://doi.org/10.1109/ARES.2010.112
29. Sametinger, J., Rozenblit, J., Lysecky, R., Ott, P.: Security challenges for medical devices. Commun. ACM **58**(4), 74–82 (2015). https://doi.org/10.1145/2667218
30. Tischer, M., Durumeric, Z., Foster, S., Duan, S., Mori, A., Bursztein, E., Bailey, M.: Users really do plug in usb drives they find. In: 2016 IEEE Symposium on Security and Privacy (SP), pp. 306–319 (2016). https://doi.org/10.1109/SP.2016.26
31. Universal Robots: Service manual—revision ur10_en_3.1.3 (2016)
32. U.S. DHS ICS-CERT: Advisory (ICSA-18-191-01). https://ics-cert.us-cert.gov/advisories/ICSA-18-191-01
33. Zanchettin, A.M., Ceriani, N.M., Rocco, P., Ding, H., Matthias, B.: Safety in human-robot collaborative manufacturing environments: metrics and control. IEEE Trans. Autom. Sci. Eng. **13**(2), 882–893 (2016). https://doi.org/10.1109/TASE.2015.2412256