TREND MICRO

In partnership with

POLITECNICO MILANO 1863
DIPARTIMENTO DI ELETTRONICA, INFORMAZIONE E BIOINGEGNERIA

Attacks on Smart Manufacturing Systems

A Forward-looking Security Analysis

Federico Maggi
Trend Micro Research

Marcello Pogliani
Politecnico di Milano

Attacks on Smart Manufacturing Systems

A Forward-looking Security Analysis

Published by
Trend Micro Research

Written by
Federico Maggi
Trend Micro Research

Marcello Pogliani
Politecnico di Milano

With contributions by
Martin Rösler, Marco Balduzzi, and Rainer Vosseler
Trend Micro Research

Stefano Zanero, Davide Quarta, and Walter Quadrini
Politecnico di Milano

Stock image used under license from Shutterstock.com

For Raimund Genes (1963-2017)

Contents

This research presents a systematic security analysis that we performed to explore a variety of attack vectors on a real smart manufacturing system and to assess the attacks that could be feasibly launched on a complex smart manufacturing system. The main, two-pronged question we want to answer is: Under which threat conditions and attacker capabilities are certain attacks possible, and what are the consequences?

While smart manufacturing systems are often isolated from other company networks, there is a trend toward less isolation between information technology (IT) and operational technology (OT) systems. This could worsen the consequences of the attacks that we describe in this research because future attackers would have more opportunities for remote entry points, which currently are relatively unlikely. Recent incidents such as the ransomware infection that halted production at a major semiconductor foundry in 2018[1] have already shown the impact of IT-to-OT lateral movement.

When traditional, well-known attacks such as those involving malware designed for general-purpose IT systems hit a smart manufacturing system, they are typically visible as unexpected or blacklisted patterns in the network or host. These cases are spotted with network and endpoint protection solutions. In this research, we take a forward-looking viewpoint and look at what future advanced attackers would do given the technology-specific attack opportunities offered by smart manufacturing environments. We consider the smart manufacturing system in its *entirety*, without focusing on the presence of specific vulnerabilities. For example: What would happen if an attacker was able to "blend in" as legitimate network traffic or expected host activity (regardless of how it could happen)? What would such an attacker do to remain persistent? Furthermore, how could the legitimate functionalities of smart manufacturing technologies be abused?

Through concrete and detailed attack descriptions, we provide insights on how to protect a real smart manufacturing system. We show, for example, how an attacker could remotely and indirectly compromise an entire system using malicious software extensions, which some vendors are already distributing via app store-like ecosystems, as the attack vector. Not only do we show that such malicious extensions have full capabilities on the target system, but we also show how an attacker could maintain a lower profile and silently use such extensions to "trojanize" the logic that will be executed on complex machines (e.g., industrial robots). We describe how the attack, once the malware is propagated in the physical machines, could continue by harvesting network and host information or planting services to support the next steps of the attack and to remain persistent.

The flourishing market of industry-grade embedded devices — often referred to as industrial internet-of-things (IoT) devices — offers another interesting entry point for attackers. Using one of these devices, we show how third-party development libraries could be abused for implementing both silent and noisy denial-of-service (DoS) attacks, e.g., to stop operations on the production floor until the device is identified and removed, thus creating additional downtime due to the missing, legitimate services that it was running.

Given the results of our research, we highlight the following security-sensitive areas in a typical smart manufacturing system:

- **Industrial software delivered as packaged add-ins, extensions, or apps** are powerful attack vectors that have not yet been seriously considered. Our findings show that if the delivery platforms (e.g., app stores) are not properly secured, they could offer a unique way to indirectly infect critical endpoints such as engineering workstations, from which attacks can propagate down to the production floor and remain persistent (e.g., in the form of machine logic introduced via a backdoor).

- **Custom industrial IoT (IIoT) devices** are gaining popularity because they allow engineers to run fully custom automation logic on the production floor, as opposed to less powerful nodes or traditional hardware such as programmable logic controllers (PLCs). We show that this flexibility and lowered access bar for developers create a change in the security management model: Instead of trusting one vendor that develops the software

running on these devices — usually the devices' vendor — the users (e.g., system integrators) will have to manage an oft-intricate chain of trust, with many third-party libraries imported in the final software. Given that attackers have recently been targeting such libraries to compromise software at its origin,[2] we deem it important to raise awareness of the very same risk in industrial settings, where it is likely to have a greater impact.

- **Human-machine interfaces (HMIs)** are a central component of the smart manufacturing ecosystem. As shown in previous research by the Trend Micro™ Zero Day Initiative™ (ZDI) program,[3] HMIs have a wide attack surface as they are general-purpose computers with many interfaces, can be seldom upgraded, and are often affected by software vulnerabilities. The complexity of HMIs is also growing. We show that current mobile HMIs suffer from the typical issues found in unsecure mobile apps — a sign that they might not be ready for widespread use. Some are deployed via sideloading, use unsecure protocols to communicate with the back-end, and are shipped with hard-coded credentials, all of which position them as one of the weak links in this complex ecosystem.

- The **manufacturing execution system (MES)** is the most sensitive endpoint in a smart manufacturing system because it acts as a trusted bridge between the production floor and the rest of the corporate network, e.g., enterprise resource planning (ERP) systems. MESs are highly customized products that revolve around one or more databases that contain complex automation logic and work plans. We show the consequences of a slight alteration in one of the databases, which could result in damaged manufactured goods if the MES was not designed with security in mind and with specific countermeasures.

- **Complex, programmable manufacturing machines** such as industrial robots possess computational power that can go beyond performing physical movements, which is their main functionality. Nowadays, they can run general-purpose computing tasks, which not only can be a source of vulnerabilities, but can also be abused by an attacker to hide malicious logic that could evade current endpoint protection solutions since it will be considered as valid machine automation code.

The impact of cyberattacks on smart manufacturing systems can be very high[4] because many organizations operating these kinds of systems are part of important industries or critical infrastructures. There are also several security challenges because an attacker can have many targets to choose from: the connection between the MES and the actuators, the sensors (the attacker can, for example, tamper with the measured values), the network itself (traffic is often unencrypted), the HMIs (e.g., traditional or mobile), humans in the loop, and a complex software supply chain with many dependencies.

Major manufacturers around the world either already use and rely on smart manufacturing technologies or will do so in the near future, and thus should consider the findings of this research.

01 Introduction

Smart manufacturing systems can be seen as the modern implementation of the totally integrated automation (TIA) concept that has been developed by Siemens since 1996.[5] But such is the complexity of smart manufacturing systems that it is difficult, if not futile, to provide any "crisp" definition of them. From a security research standpoint, it is challenging to obtain access to a sufficiently generic, fully functioning system, deployed within realistic conditions, because the concept of a "generic" or "reference" smart manufacturing system does not really exist. Therefore, any security analysis — including this one — must be interpreted with a grain of salt: It is easy to jump to conclusions such as, "All smart manufacturing systems are unsecure," or, worse, to view attack scenarios as ready-to-use best practices on "how to secure smart manufacturing systems." In this research, our aim is to provide food-for-thought examples and use cases intended for concerned organizations and individuals to carefully contextualize in their specific settings.

In this section, we describe the scope of our analysis (an actual smart manufacturing system deployed in an Industry 4.0 research laboratory), the methodology we used during our research (a holistic, hands-on driven approach), the research angle we employed (focusing on concrete attack vectors in the hands of a forward-looking attacker), and some background concepts needed throughout the remainder of this research paper.

1.1 Scope

Tracing a boundary for this research was difficult because smart manufacturing encompasses a wide and diverse set of technologies and disciplines. Ultimately, we decided to concentrate our attention on a single instance of a smart manufacturing system, including the software ecosystem that revolves around it.

Figure 1. A photo of Industry 4.0 Lab, the system that we analyzed during this research

The system under analysis is Industry 4.0 Lab,[6] a research laboratory assembled by Festo and currently housed at the School of Management of Politecnico di Milano,[7] the largest technical university in Italy. Although the lab, which costs around €250,000, is used for research and education purposes, it is engineered with the same equipment and basic principles used on real-world production floors.

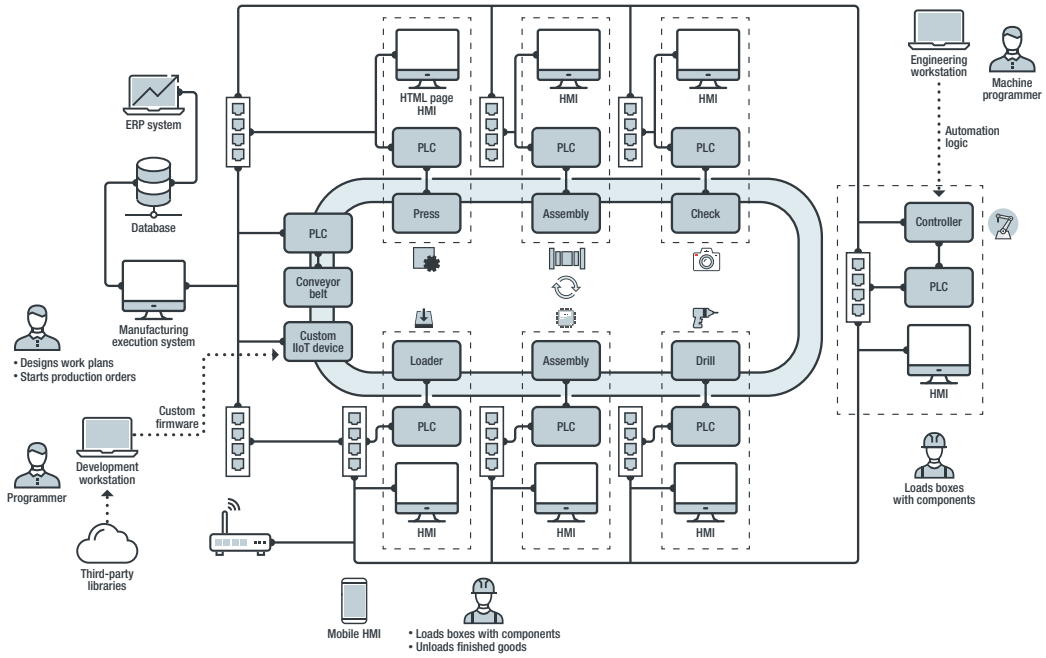


Figure 2. The overall architecture of the smart manufacturing plant that we used in this research
Note: A detailed description of the system and its parts is in Section 2.

Although we did consider their existence within the ecosystem, we left human resources (e.g., operators, contractors, engineers) out of our security analysis since the extent to which human behavior directly affects the operational security of a smart manufacturing system was beyond our scope. However, we considered the opposite — how the technology could be abused with the goal of influencing the decisions of the operator, e.g., via user interface (UI) tricks — but did not examine whether and how a human would successfully fall for such tricks.

1.2 Methodology

We followed a practical, hands-on methodology: We invested time to understand the internals of the smart manufacturing system, while also using it to produce some goods, as a real operator would do. Once we understood the basic operational aspects, we consulted with the lab's plant maintainers to better understand the details of the system and also read technical documentation as needed.

Upon gaining a deep enough understanding of what we were working with, we started thinking about the security models at play, under what threat model assumptions they could be violated, and how. If we found a feasible example, we tried to implement an attack to prove our point. For instance, once we discovered that the communication between the programmable logic controllers (PLCs), the manufacturing execution system (MES), and the human-machine interfaces (HMIs) was unencrypted, we verified whether valid messages could be spoofed, and to what extent these would be accepted by the endpoints. We always tried to verify our findings in an end-to-end fashion, i.e., without limiting them to the attack possibilities. Overall, we created proofs of concept for five attacks, which are described in Section 4.

We consulted several times with operational technology (OT) experts, including automation engineers, operators, and industrial robot programmers. On top of that, we made extensive use of online discussion forums typically used by experts to exchange tips and best practices. Later on, we used the very same forums to recruit some volunteers for an online survey, and to "measure" the overall security awareness of the community, the results of which are discussed in Section 2.2.

1.3 Angle

We looked at what an advanced attacker would be able to accomplish, given the technology-specific attack opportunities afforded by smart manufacturing environments. Taken on its own, each part of a smart manufacturing system likely would have already been analyzed from a security viewpoint, either by us or by other researchers; industrial robots, PLCs, HMIs, industrial endpoints or networking appliances, and the like have all been subject to scrutiny. Thus, we focused instead on how an advanced (and creative) attacker, with access to one or more of the system's components, would be able find their way into other parts of the system. In other words, this research focuses on the system in its *entirety* as a set of entry points and targets, under various attacker model assumptions.

Despite discovering and reporting some vulnerabilities, we stress that this analysis does not focus on any specific vulnerabilities, new or existing, in the products of the named vendors. Rather, it focuses on attack vectors, design issues, and post-exploitation opportunities. As a matter of fact, we took advantage of the physical laboratory as a concrete starting point to understand the architecture of a real smart manufacturing system and perform practical experiments on it; we did not use it solely as an exploitation target. When needed, we responsibly disclosed any security vulnerabilities that we discovered along the way through the Trend Micro™ Zero Day Initiative™ (ZDI) program.

1.4 Background

We assume that readers of this paper are familiar with the concepts of industrial automation that are essential to understanding the high-level functionalities and the interactions of components at various levels of the automation pyramid,[8] as depicted in Figure 3. Section 2 introduces the basic terminology and provides a technical overview of a typical smart manufacturing system.

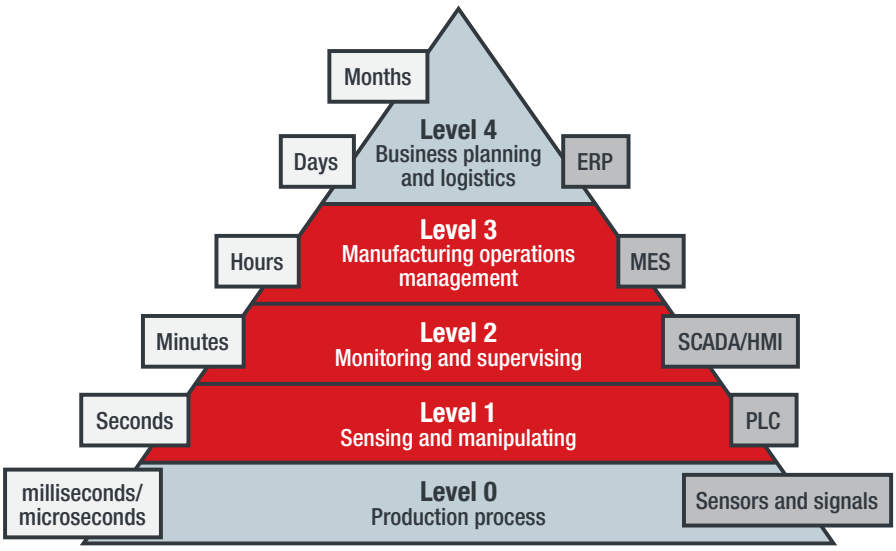


Figure 3. The automation pyramid

This research focuses on Levels 1 to 3. Specifically, we assume that the sensors are trusted components, so an attacker may be able to spoof signals coming from the real sensors (without compromising the actual sensors). Similarly, we assume that Level 4 is considered trusted by the lower layers, which means that an attacker may be able to craft messages coming from it (although we did not investigate how that could happen).

1.5 Research Questions

We collected a list of "research questions" that we kept as a reference during our work. It is not meant to be exhaustive, but we hope that it will be a useful resource for future OT security research:

- Under which threat and attacker models are certain attacks possible, and what are the consequences? (This is the main question.)
- Are there any overlooked vectors that could facilitate an attacker's getting a foothold in these systems?
- What is the security impact of modern industrial software development practices, including the use of open libraries, with complex interdependencies?
- What is the cybersecurity awareness level of the technical personnel who engineer, program, and operate in smart manufacturing environments?

Readers who wish to know our high-level answers to these questions without going through the rest of this paper may do so by reading our conclusion in Section 6.

02 Technology

Smart manufacturing is the convergence of automation and information technology (IT), and is at the essence of the ongoing industrial revolution known as Industry 4.0.[9] To draw an analogy: In the past, we would have needed to purchase a hardware computer in order to run our software, but today, we can quickly provision a bare-metal server in a data center and connect to it in less than five minutes — thanks to the integration and automation of IT operations. Similarly, we are getting to a point where manufacturing a product will be completely streamlined, with little or no human intervention required.

Smart manufacturing has become a major technology trend. For example, 10% of the 2021 edition of Hannover Messe, one of the largest industry exhibitions in the world,[10] will be dedicated to digital factories (based on the number of exhibitors in the different categories), with more than 600 exhibitors, about 300 of which exclusively target the manufacturing industry. Within the manufacturing industry, smart manufacturing affects sectors including those that deal with: minerals; metals; oil; food; chemicals; textiles and clothing; electric products and electronics; furniture; glass and ceramics; leather, rubber and plastics; paper, cardboard, and related products; pharmaceutical products; tobacco; and cars and automotive products, components, and equipment.

In this section, we describe the technologies that drive a smart manufacturing system by focusing on the features that are relevant from a security standpoint. We conclude this section with a quantitative analysis of the security awareness of the online communities of practitioners who both talk about and work in the fields that revolve around smart manufacturing technologies.

2.1 Machines and Process

Industry 4.0 Lab, the smart manufacturing system that we analyzed during this research, comprises seven stations, each with PLCs and HMIs made by Siemens, various physical actuators (e.g., drills, presses) and sensors (e.g., temperature, pressure, camera), a conveyor belt, and a Mitsubishi Melfa industrial robot. Industry 4.0 Lab is part of the Fenix Project,[11] which is partially funded by the Horizon 2020 research and innovation program of the European Union (EU).[12] The goal of the Fenix project is to create a reconfigurable multi-material pilot plant producing various goods, including but not limited to 3D printing metal powders, customized jewels, and advanced filaments. In its current installation, the system produces toy cell phones.

Figure 4. Details of a machine (drill), HMI, and PLC in the target manufacturing system

The target smart manufacturing system (informally referred to as "the carousel" because of its circular workflow) is made up of stations, each comprising at least three parts, as pictured in Figure 4:

- A physical machine, such as a loader, drill, press, camera, or industrial robot, which does the actual work.
- An HMI, which is used by the operator for monitoring and controlling the progress.
- A PLC, which serves as the interface for the interaction between the machine, the HMI, and the rest of the network.

The PLCs are Siemens Simatic DP CPU 1510SP-1 PN units,[13] the HMIs are Siemens Simatic HMI TP700 Comfort units,[14] the industrial robot is a Mitsubishi Melfa V-2AJ,[15] and the network is made and managed out of Siemens Scalance X208 switches.[16]

Despite discovering and reporting some vulnerabilities during our analysis, we stress that *we do not focus on exploiting or researching specific vulnerabilities (either new or existing) in the products of the named vendors* — they are reported merely for completeness. Rather, we focus on attack vectors, design issues, and post-exploitation opportunities.

2.1.1 Manufacturing Execution System

The production process is coordinated by the manufacturing execution system (MES), which is a complex logic layer on top of a database. The MES is the interface between the enterprise resource planning (ERP) system, if any, and the actual physical plant. If used by itself, the MES can also incorporate some ERP system-like functionalities. The MES allows the writing of "recipes" that specify the production steps. These can be seen as "work templates" (i.e., sequences of generic actions), which are translated to network packets to the various stations (i.e., the PLC and HMI) once they receive input parameters (e.g., the number and variant of items to produce). When in execution, the MES receives feedback from the stations, ensures that the sequence of instructions in the work order is followed, and takes corrective actions if needed. Tracking each part is easy because each pallet is always transported by a carrier identified by a radio frequency identification (RFID) tag, which is known to the MES.

2.1.2 Roles

We consider three broad roles in a smart manufacturing system:

- The system operator, who supervises the production (e.g., through HMIs), loads supplies as needed, checks for alerts, and unloads the finished goods if needed.

- The MES operator, who designs work templates and interacts with the MES.
- The machine engineer or programmer, who designs and integrates the low-level logic in specific machines (e.g., robots, drills) and translates high-level order commands (e.g., "drill here," "move arm 30 degrees left") into actuator instructions.

While some of these roles can be filled by employees of the factory, others can be filled by outsiders (e.g., consultants, system integrators, and other contractors). For instance, the system operator can be an employee, the MES operator can be a system integrator, and the machine engineer can be a contractor who is called upon when needed (e.g., when a new task program needs to be developed for a new machine or when maintenance of existing logic is required). Therefore, some workstations are not always connected to the network, and the computers, even if never connected, may not be under the complete control of the factory's IT staff. Consultants may use their own computers to develop automation logic code, which is then transferred to the factory floor (e.g., via portable media), without connecting their computers to the factory network. This not only makes security management more complex, but it also allows for the creation of new attack opportunities, as summarized in Figure 5 and described in Section 4. Outsourcing, especially in low-wage countries, also includes the risk of bribery or extortion of contractors, which could be considered by attackers.

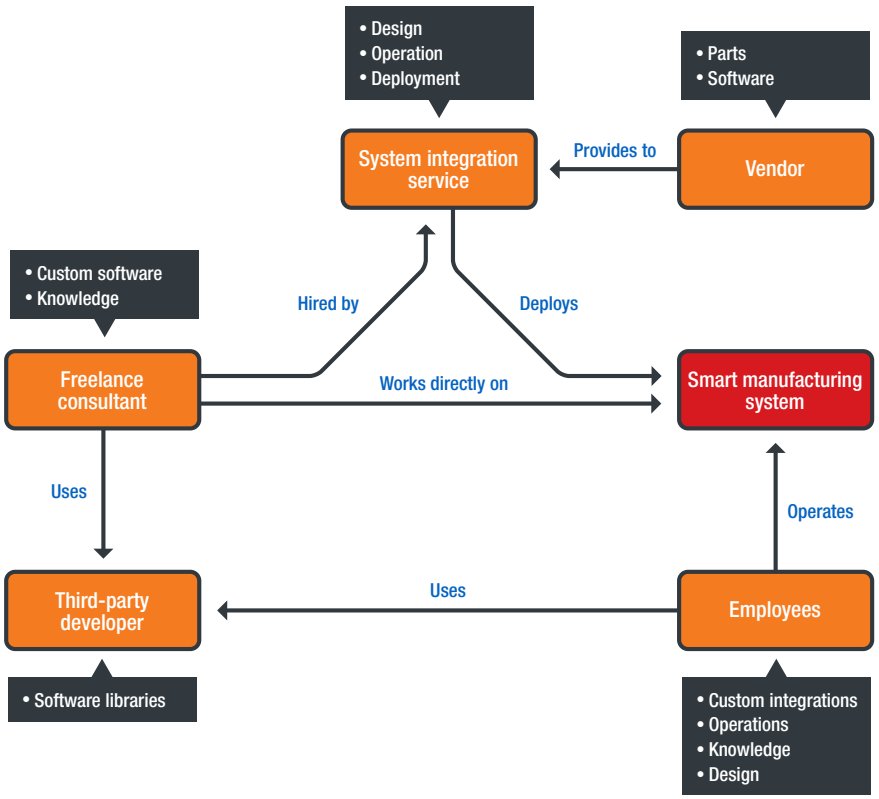


Figure 5. The various roles (orange) that supply software, parts, and expertise to a smart manufacturing plant (red), and the relationships between them (blue)

2.1.3 Production Flow

The system operator interacts with the system mainly through the HMI, which we assume is considered trusted by the operator.

Figure 6. Close-up shots of the various stations of the smart manufacturing system

In the case of the system that we analyzed, the typical production workflow is as follows:

1. The operator ensures that there are enough input parts and starts the production from the HMI on the first station.
2. The first station loads the first half of the toy phone case from a magazine onto the conveyor belt.
3. The second station assembles the case of the toy cell phone.
4. A drill drills a (programmable) number of holes through the phone case.
5. The industrial robot places the printed circuit board (PCB) into the case and assembles the electronic components.
6. The half-assembled phone goes through an intermediate visual check with a camera.
7. The top case is assembled.
8. A press pushes the top case down.
9. The conveyor belt ensures that the finished product goes back to the loader station and informs the MES to display a message on the HMI, prompting the operator to pick it up.

2.1.4 Mobile HMIs

We complemented the smart manufacturing system by considering mobile HMIs. As discussed in Section 3.3.3.2, mobile HMIs are gaining traction in the manufacturing world because they allow operators to conveniently interact with the system, receive production feedback, and even control machines (e.g., robots) with high precision and with the same level of control offered by classic manual interfaces (e.g., joysticks, physical buttons). However, there are concerns that are holding back the ubiquitous adoption of mobile HMIs, including issues pertaining to integration, risk management, and, of course, cybersecurity.[17]

2.2 Voices From Field Experts

There is a common belief that orthogonal disciplines (e.g., hardware or software engineering, automation) have just started to fully embrace the importance of cybersecurity. But this is often backed only by anecdotes. As part of our research, we were interested in understanding how and how often “security matters” are discussed in OT and industrial automation online communities.

2.2.1 Talking 'Security'

After compiling a list of 11 OT-related online forums, we collected some analytics data to gain insights into the activity levels: the number of users, the topics and replies, and the oldest posts. Then, combining the forums' search feature with site-specific web searches, we counted how many times security-related words were used in the discussions. This clearly does not provide an unbiased outcome. The keyword "security," for example, may or may not always indicate that a post is actually about security (e.g., a phrase or sentence like, "Please update your software to the most recent release to get all the security fixes," does not necessarily indicate that the users are talking about a security topic). However, we accept and interpret these results as an upper bound — an optimistic snapshot of the true situation.

Community	Affiliation	Indexed since	Total number of users	Total number of topics	Total number of replies or comments	Overall number of mentions of security-related terms
Control.com[18]	N/A	1997	N/A	N/A	69,700	**5,068**
PLC.MyForum.ro[19]	N/A	2012	93,948	41,841	N/A	**1,968**
Mr.PLC[20]	N/A	2006	46,144	33,540	164,787	**1,810**
Robotforum[21]	Robtec	2006	17,611	19,166	90,134	**892**
Reddit - robotics[22]	N/A	2008	83,614	N/A	N/A	**638**
Adam Forum[23]	N/A	2010	33,286	3,783	6,702	**170**
Automation Forum[24]	N/A	2012	220	1,900	7,800	**147**
DoF[25]	Robotiq	2016	N/A	1,500	N/A	**83**
ABB Robotics[26]	ABB	2013	19,723	8,959	19,723	**68**
Universal Robots[27]	Universal Robots	2017	N/A	N/A	N/A	**24**
SolisPLC[28]	SolisPLC	2018	134	36	87	**0**

Table 1. The OT-related online forums we considered and their relevant details (as of August 2019)

As expected, the conversations that mentioned security-related keywords (e.g., "security," "vulnerability," "hacked," "attack," and variations thereof) were a slim minority. There have been anecdotal claims that people from the OT and IT security community "talk different languages" and "have different priorities," and the numbers in Table 1 provide concrete evidence in support of these claims.

The lack of discussion regarding security lends credence to the belief that many of these communities often view "security" as mere compliance with whatever regulations apply to their specific manufacturing fields. This, of course, is a very far cry from the desired situation, where people in the industry fully embrace security concepts and design as part of their work. This further motivates our research, with the goal of raising awareness specifically within these communities.

2.2.2 Asking Experts Technical Questions

The thinking that industrial cybersecurity is a major priority and industrial systems are likely to be a target of cyberattacks is not new. However, we were interested in understanding the technical aspects that might create preconditions for increased risk. For example, while it was interesting to know whether a manager was aware of the possibility of cyberattacks that could compromise the safety of their

organization's equipment, we instead wanted to know whether the engineers used custom software, and who created it and why. Indeed, assuming "perfect" network-level protection, there is very little that can be done when a trusted yet malicious or vulnerable piece of software manages to sneak into industrial plants.

We interviewed six experts over the phone and had them fill in a fairly technical questionnaire. We focused on select experts within our trusted contacts, ranging from consultants working for system integrators to employees of major system integrators and even academic professors. On top of that, we reached out to 14 OT practitioners through online community discussion groups and mailing lists. Overall, we had the six experts helping us at various points throughout our research and 20 respondents for our survey; 70% of them were from the industry, 10% were from academia, and 20% were system integrators.

One of the key takeaways, as indicated in Figure 7, was that almost half of the respondents confirmed the use of custom industrial internet-of-things (IIoT) devices (e.g., Arduino, Raspberry Pi) on their factory floor; custom code was developed mostly internally, with some parts outsourced — thus trusting the external developers — with only a minority having a risk assessment process in place for these devices. Another was that industrial robot programming languages were used by more than half of the respondents, who confirmed that most of their use cases were interconnection with the external world — in some cases by using advanced features. We clarify some of these aspects in Sections 3.3.2, 3.3.5, 4.1, and 4.5.

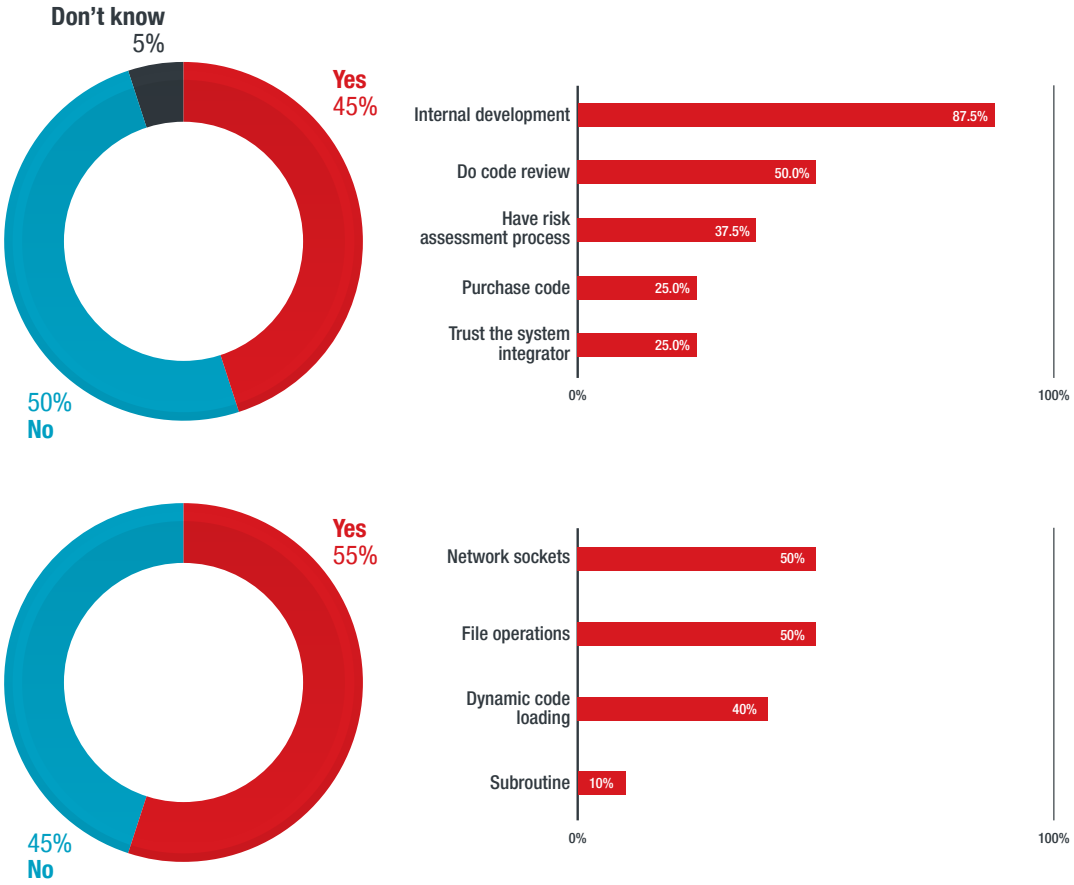


Figure 7. The main answers to our survey, focusing on the presence of custom IIoT devices and software, with some hints on development practices

03 Security Analysis

In a smart manufacturing system, traditional attacks are visible as unexpected or blacklisted patterns in the network or host activity, and can be spotted and blocked with current countermeasures, such as network and endpoint protection solutions. We do not consider these cases as a subject of this research.

We take a different viewpoint and look at what an advanced attacker[29] would be able to accomplish, while considering the system as a whole, made up of more than just what exists on the factory floor. What would happen, for example, if an attacker was able to blend in as legitimate network traffic or normal host activity? What would such an attacker do to remain persistent? Are there some unique attack opportunities, perhaps outside the network perimeter, which are currently being overlooked?

Figure 8 provides a visualization of the many dependencies in the software and data ecosystem that revolves around a smart manufacturing system. At the development stage, we see software add-ins and digital twins being supplied by the vendor or being developed on the engineering workstation (and optionally uploaded to an online catalog, usually provided by the software extension). This workstation is also used to create custom automation logic (for machines like robots) or firmware for custom IIoT devices. All of these, together with the other components like the HMI, MES, and PLC, make the automation logic work. High-level business decisions are translated into data written in the ERP system (or some other database), which in turn determines the actions scheduled by the MES, which then defines the automation routines executed by the PLC. In this, we can see the indirect impact of the software supply chain in the final automation actions (as depicted in the lower left part of Figure 8).

Figure 9 shows a visualization of the attack opportunities in the data and software dependencies. The attack involving compromise through a malicious industrial add-in and the attack involving “trojanization” of a custom IIoT device (described in Sections 4.1 and 4.2, respectively) abuse software components, which is possible (and is already being done in the wild) nowadays because of the complex supply chain, which in turn contains plenty of weak points. The attack involving a vulnerable mobile HMI (described in Section 4.3) shows how leaked information in a mobile HMI can be exploited to gain access to the machine controlled by that HMI (in our case, an industrial robot). The attack involving data mangling on the MES (described in Section 4.4) shows how any manipulation of data at the ERP system or database level can have a later impact on the automation. The attack involving the vulnerable or malicious automation logic in a complex manufacturing machine (described in Section 4.5) is by nature more sophisticated, because it exploits weaknesses in the automation logic.

From our conversations with domain experts (introduced in Section 2.2), it emerged, among other things, that isolating smart manufacturing systems within a dedicated, isolated network is common practice. We also understood that these systems are treated like black boxes, in the sense that it is assumed

that nobody will ever be able to compromise them. On the other hand, connectivity is increasing and vendors are pushing for wireless networks on the factory floor, with assets such as industrial robots directly connected to them.

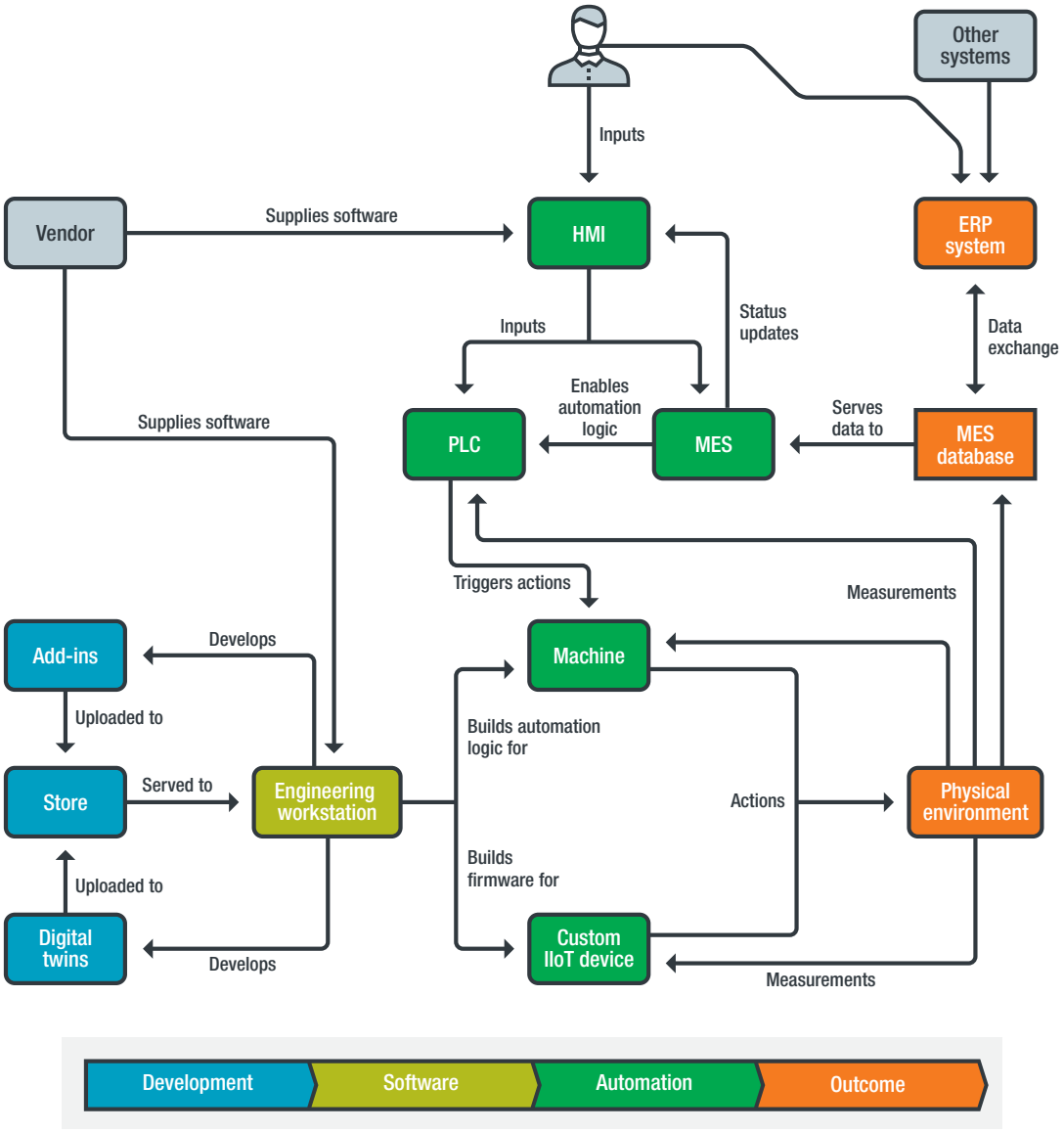


Figure 8. Data and software dependencies in the context of a smart manufacturing system

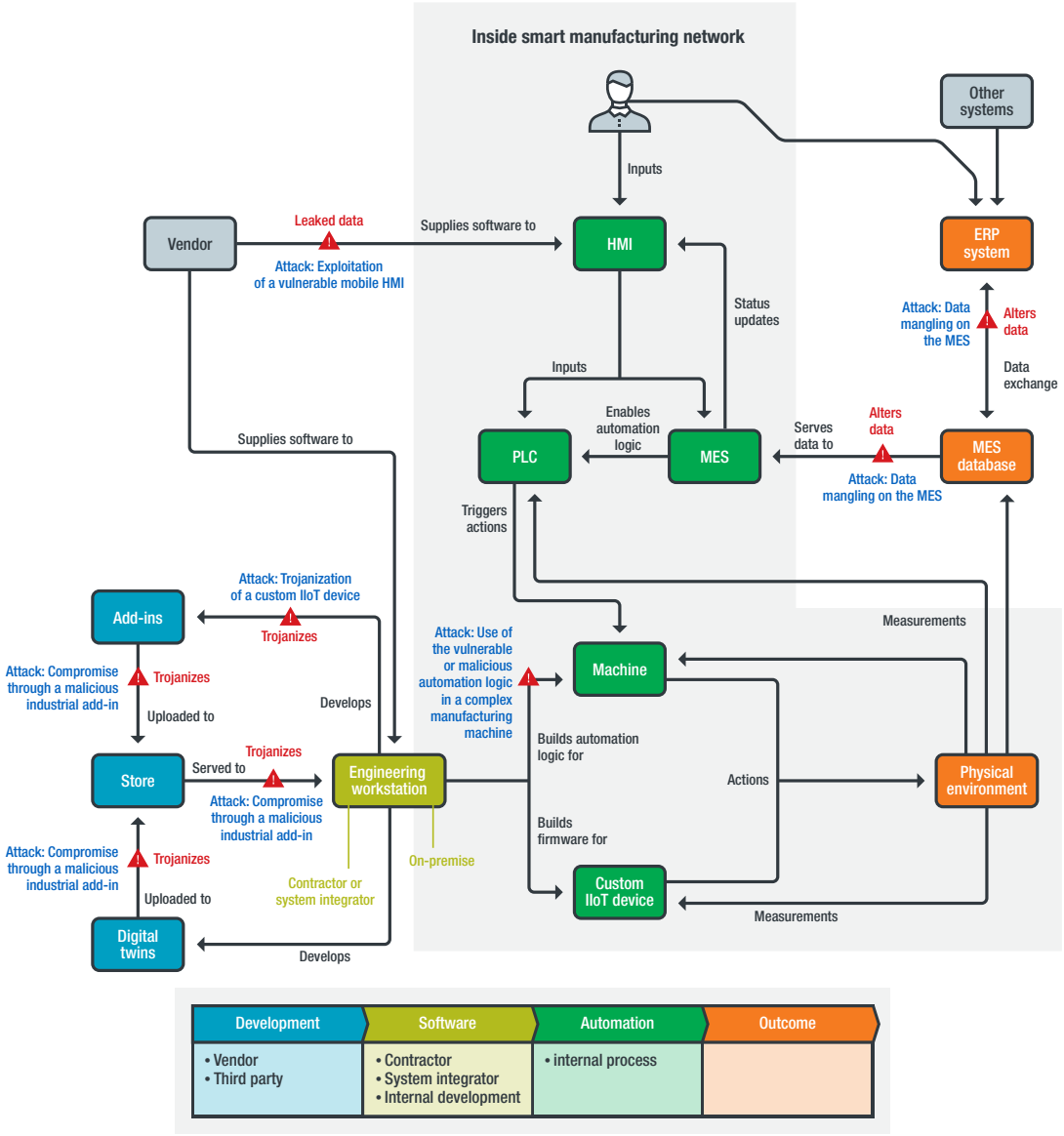


Figure 9. Attack opportunities in the data and software dependencies

3.1 Attacker Goals, Resources, and Capabilities

In an industrial setting, we are looking at advanced attackers with enough resources and capabilities to compromise at least one machine, directly or indirectly connected to the smart manufacturing system. We consider attackers who want to cause malfunctions, damage the produced goods, or alter the workflow such that it would manufacture defective products. There may be a variety of motives: Attackers may be employed by a competitor, may be financially motivated (e.g., attackers may request payment in exchange for revealing details of the batches in which they have introduced "hidden" defects, as we pointed out in one of our previous researches[30]), or may even just want to affect the factory's overall reputation. Attackers may also be interested in the automation logic, which is usually a well-guarded intellectual property.

Depending on their profiles, attackers may or may not have a foothold in the smart manufacturing system network. For example, if an attacker is on an adjacent network, they may be able to write one field in the database of the MES because the MES is often a "bridge" between the enterprise and the factory floor networks. A remote attacker with no access to the factory network may attempt to create a malicious program for the industrial robot and use an insider or other software-based attacks to make it run on the robot. If the attacker knows that the system integrator or the target factory is using a specific piece of development software, they will likely target either the software itself or some third-party extensions for that software.

In the past several years, we observed a number of supply chain attacks on software development tools or libraries, especially open-source ones.[31, 32, 33, 34, 35] Interestingly, 42% of attacks on the manufacturing industry reportedly do not directly target facilities, but rather target some of the systems along the supply chain.[36]

3.2 Entry Points

This section provides a security-oriented overview of a smart manufacturing system, from which we derive its attack surface or the set of entry points that an attacker may consider targeting.

Figure 10 shows the security-sensitive areas of the smart manufacturing system that we analyzed, with the physical network perimeter highlighted. In our case, this indicates where the factory floor network is separated from other networks (e.g., internet, enterprise network) by firewalls. The red indicators signify endpoints that can be used as entry points for attacks.

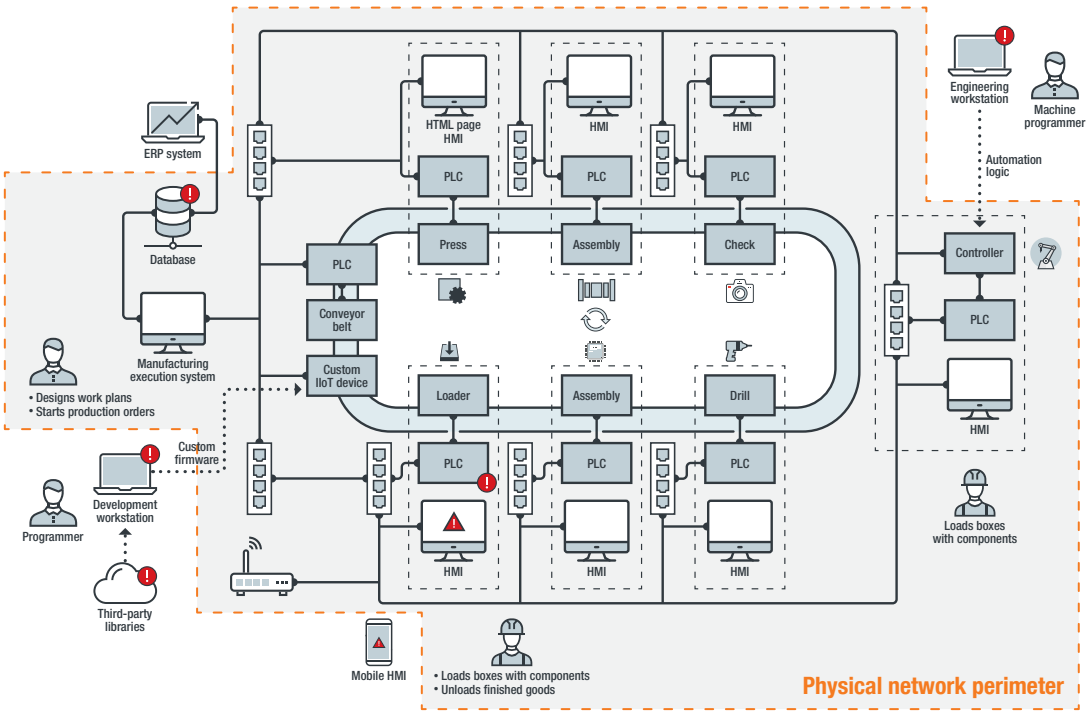


Figure 10. An overview of the attack surface, with the physical network perimeter highlighted

3.2.1 Engineering Workstation

The engineering workstation is a shared system with domain users that is always connected to the production floor. It is used to develop and deploy program logic, or to connect to field devices (e.g., PLCs, HMIs) for maintenance, diagnostics, or reprogramming. Occasionally, it is used to simply deploy programs that are developed elsewhere, perhaps outside the factory premises by personnel working for the system integrator.

There is a trust relation between any workstation used for engineering purposes and the rest of the system. Sometimes this relation is known and is part of the security planning. Other times, it becomes more difficult to see, considering how many indirect or implicit trust relations there are between, on one side, the person who develops automation logic and, on the other, the smart manufacturing system where the logic is finally deployed. This does not necessarily mean that the developer is malicious: Their computer could simply have been compromised, or even one library that they use could have been compromised at the source. A case in point is the XcodeGhost malware, which was used in one of the earliest instances of supply chain attacks: One of the techniques of the malware was to modify the Xcode compiler such that the compiled iOS apps would be infected.[37] As we demonstrate in Section 4.1, the industrial software that we analyzed offers concrete opportunities for attackers to compromise the entire engineering workstation or alter the digital twins.

3.2.2 Custom IIoT Device Development Environment

There are custom IIoT devices (e.g., embedded systems, Arduino-like devices, Raspberry Pi, or other single-board computers) programmed by system integrators or internal employees. They are gaining popularity because they allow more automation flexibility than classic automation hardware such as PLCs.

There are many trust relations between the multiple software libraries in this ecosystem and the smart manufacturing system, where the final software is deployed. What we discuss in the previous section regarding the engineering workstation applies even more strongly in this instance: It is highly likely that a developer needs to use a third-party library, a library based on a third-party library, or a third-party library based on a library from another party. This software dependency chain is very complex. As we show in Section 4.2, there is no way for a developer to easily validate the end-to-end integrity and authenticity of a library, which can lead to the inclusion of trojanized components.

3.2.3 MES Database

The MES database is often shared with the upper layers of the automation pyramid. Its function is to contain work orders and work templates, which are clearly sensitive data. When a work template is created on the MES, a new record is saved to the database. Similarly, when a work order is started, the state of the production operations is updated in the database.

At a conceptual level, the MES trusts the data coming from the database. This implies that if there is no authentication and integrity of the database storage, an attacker on the network (or on the database) could forge or alter records, thus resulting in altered production. As we show in Section 4.4, alteration can happen at the product-feature level and can be nondestructive.

3.3 Targets

Taking a more in-depth look at the peculiar aspects of smart manufacturing technologies, this section highlights the components that offer attack opportunities.

3.3.1 Industrial Add-ins

The delivery mechanism of industrial software is evolving to keep up with the pace of innovation. Specifically, we found out that some solutions are being inspired by app-store models. For example, ABB has an app store[38] where anyone can register (registration is automatic with only email validation in place) and upload add-ins for ABB's RobotStudio, which is used by engineers to write automation logic for ABB industrial robots. There are about 1,000 add-ins on the store, some of which have been downloaded thousands of times. These numbers must be interpreted by first considering that industrial robotics is currently a niche sector, where developers tend to keep everything "in house." This is expected to change, and app stores such as ABB's are the first signs of this change in direction.

Intelligent Plant's solution[39] is similar in spirit to ABB's app store. It is not dedicated to robotics in particular, however, but to industrial applications in general. Individuals cannot upload apps; only registered businesses can. After trying to contact the owners, we argued that the apps are procured via a business-to-business (B2B) channel and served via a business-to-consumer (B2C) channel.

OrangeApps[40] is dedicated to Kuka, despite not being developed or managed by Kuka directly. Unlike ABB's app store, OrangeApps does not accept submissions from users; it is seemingly a closed ecosystem. Interestingly, it serves apps for both desktop software and the industrial robot controller. Therefore, the apps obtained from OrangeApps include code that runs directly on the industrial robot controller. Unfortunately, we noticed that the network transport uses plain HTTP, which opens the possibility for network man-in-the-middle (MitM) attacks.

Siwiat's solution[41] has a slightly different model: The vendor provides the hardware (an IIoT device), which has functionalities that can be extended by downloading apps from its app store. This software delivery model is very similar to the mobile device ecosystem, where users just purchase a piece of hardware and expand its functionality by downloading apps delivered from (trusted) stores. As a matter of fact, anything that comes from the app store is considered trusted.

3.3.1.1 Security Trade-off

This centralization provides some benefits, as we have learned from mobile app stores. However, we argue that in their current state, these models require some attention. Software extensions or apps can become weak spots, and app stores can become interesting targets for attackers. Browser extensions and mobile apps have highlighted the importance of sandboxing and the crucial role of app stores, which should have solid vetting procedures (e.g., modern app stores have continuous scans for any new uploaded app). Early browser extensions and mobile apps allowed full system access, but the advent of app stores changed this due to the need for more streamlined software delivery. The most important security change is that browser extensions and mobile apps are now sandboxed, as they should be, because they are essentially trusted software running on a computer. To access the network, they need to explicitly declare (and be granted) specific permissions. Sensitive software like industrial engineering software needs to embrace the same sandboxing principles and limit the scope of the runtime.

3.3.1.2 Lack of Sandboxing

RobotStudio, an offline programming (OLP) application for desktop computers, is used by engineers to write automation logic for ABB industrial robots. It allows for the creation of interactive 3D digital twins of an industrial robot deployment, on top of which the automation logic code is written and tested.

RobotStudio, like other OLPs, comes with a plug-in or add-in system, which allows the extension of its functionality by loading dynamic link libraries (DLLs) in proper locations. Such add-ins are accompanied by Extensible Markup Language (XML) metadata for proper packaging, distribution, and loading. Essentially, when a DLL is loaded by the main software, the machine code in the DLL is executed. There is no isolation system or privilege separation: Add-ins possess "full" system and network access, to the same extent of a regular process running with RobotStudio's privileges.

The code written using design environments such as this ends up on the factory floor. Therefore, an attacker who is able to compromise a plug-in or add-in can in turn obtain indirect access to the automation code that is being written — and that will run the automation logic.

An attack that exploits a malicious industrial add-in to propagate in a smart manufacturing system is described in Section 4.1.

3.3.2 Custom IIoT Devices

There is a wide and highly fragmented offering of "industry-grade" embedded devices (often advertised as "Arduino-compatible") that are used for both rapid prototyping and production use. Arduino Industrial 101,[42] Industrial Shields devices,[43] Industrino,[44] Iono Arduino,[45] and Siemens Simatic IoT2000[46] are just a few examples. These IIoT devices bring full software customization capabilities in the hands of end users. But we argue that the growing need for customization and flexibility increases the attack surface of an industrial plant such as a smart manufacturing system.

Figure 11. Examples of industry-grade IoT devices: Iono Arduino (left) and a device from Industrial Shields (right)

3.3.2.1 Widespread Use

Most of the domain experts whom we interviewed confirmed that they used several custom devices, running custom firmware, on real-world production floors. Even in our Industry 4.0 Lab environment, there is a (separate) network of Raspberry Pi nodes that monitors the physical conditions of the plant using sensors (e.g., temperature, pressure, light, noise). Raspberry Pi devices can run anything, including Linux malware — and the same is true for Arduino or Arduino-compatible boards.[47, 48]

3.3.2.2 Malicious Devices and Hardware Implants

"Extra" devices plugged to the floor network are becoming common, representing a risk on their own. Indeed, there have been cases where such devices have flown under the radar and been used to break into critical facilities. The case of the NASA Jet Propulsion Laboratory (JPL) is a prominent example: In 2018, a hacker accessed the JPL network by targeting a Raspberry Pi device that was not authorized to be connected to it.[49] Moreover, the miniaturization of electronic components and the increased accessibility of fabrication laboratories make it possible to create hardware implants that are as small as the metal portion of a USB key.[50]

3.3.2.3 Complex Software Supply Chain

Unlike devices such as classic PLCs, which run a simple loop and are bound to a rather simple execution model, custom IIoT devices can run complex firmware and often include several external libraries, with further dependencies. In short, the software supply chain of custom IIoT devices is more complex to manage than that of a vendor-supplied hardware and software solution. The main risk is that apart from official libraries shipped by Arduino, there is no integrity mechanism that can guarantee that the libraries used by these devices are authentic and have not been tampered with. Indeed, attackers have realized that they could compromise a high number of machines at once if they could get to the "source," i.e., either by compromising a popular library or by "typosquatting" to have their code included in the final product, instead of the original one.

An attack that uses a trojanized software library to cause malfunctions in a smart manufacturing system is described in Section 4.2.

3.3.3 HMIs

There have been numerous security analyses of traditional HMI software, such as the research conducted by the ZDI in 2017,[51] which looked at the state of HMIs and highlighted that they often run outdated, vulnerable software. In this paper, we emphasize that HMI technology and custom deployments create opportunities for types of attacks beyond the classic exploitation of software vulnerabilities on the HMI side.

3.3.3.1 Traditional HMIs

Web- and cloud-based solutions, as well as app- or plug-in-based systems, have led to traditional HMIs' becoming more interconnected. HMIs have also evolved from a statically defined concept of "interaction" to a more flexible one, providing the means for end users to design or customize interfaces and quickly upload and integrate them into existing systems. These peculiarities make HMI components complex, leading in turn to a larger attack surface.

For example, even in the test setup of the research lab that we used, the HMIs are hybrid and use an embedded web browser that allows the factory operators to customize the UI (e.g., by serving custom HTML or JavaScript resources) without the need for system integrator intervention. Our domain experts working on the engineering of manufacturing plants for large customers confirmed that this was frequently requested by clients. We verified that, as shown in Figure 12, an attacker can manipulate a simple webpage not only to convey exploits but also, in the absence of vulnerabilities on the HMI side, to play several UI tricks to fool the operator and influence their decisions (e.g., simulate errors or

emergencies). A user is likely to trust the HMI screen and act based on its input, especially if there is no way to ensure the authenticity of the embedded webpage.

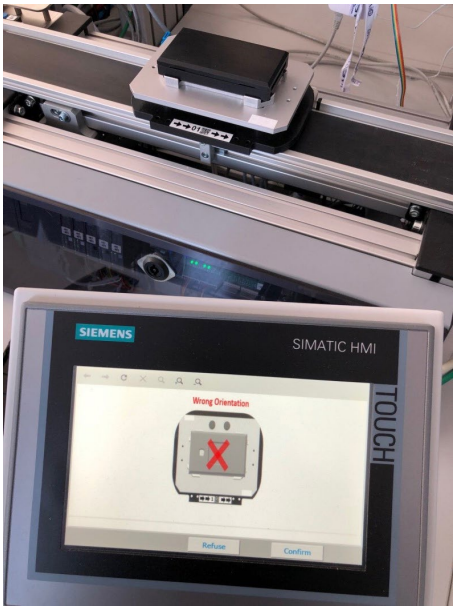


Figure 12. UI manipulation attack on the HMI: While the manufactured item is finished correctly, the HMI displays an error.

3.3.3.2 Mobile HMIs

Because of their flexibility and ease of use, mobile devices such as tablets and smartphones make for good HMIs. Despite HMIs' being a relatively niche market, we found over 170 HMI apps on the Google Play Store, more than 40 of which had over 1,000 installations — even up to more than 100,000 in some cases. Not only do we foresee a growing demand for these solutions, but we also see a usability and security benefit of mobile HMIs versus classic ones. In addition to the fact that users have grown accustomed to using mobile devices, they are also far more flexible and easier to manage than the industrial computers that run HMI software. They are easier still to keep updated and are inherently more secure since apps running on modern mobile operating systems are sandboxed — to name one important feature that is lacking in touch-based industrial computers running (outdated versions of) Microsoft Windows.

However, because of their flexibility, and because the hardware is a general-purpose computer, mobile HMIs are subject to other classes of attacks. At the physical network layer, mobile HMIs are connected via wireless protocols (Wi-Fi or Bluetooth), which make them more accessible to an attacker within range than a wired connection. The main risk is that mobile HMIs are designed with the same assumptions as traditional HMIs, i.e., that they are in a closed, wired network. For example, in Comau's PickApp, an HMI used to interact with industrial robots, the network protocols used do not enforce any integrity or confidentiality of the data, nor do they authenticate or perform any attestation of the endpoint, which means that they will trust any data as long as it complies with their application protocol. This is discussed in more detail in Section 4.3.

One important point is that, despite being sandboxed, mobile HMI apps exist with other apps and can interact with other software (e.g., via Android intents). The main risk of this is that other apps installed on a device are trusted but are not necessarily trustworthy. On Android, it is indeed possible to invoke certain actions of an app from another app via the so-called intent mechanism. Therefore, a malicious

app, even if sandboxed and isolated from the target mobile HMI, could invoke certain actions (e.g., the click of a soft button) on the mobile HMI, which would indirectly affect the physical machine it controls (e.g., move the machine). Mobile HMI apps are thus in a very "powerful" position, because any command that they send will be trusted by the machine, making them in turn a very interesting target. A similar reasoning can be done regarding the use of external storage in mobile apps, which exposes them to cross-app data leaks because the external storage is shared among all apps.

Mobile HMI apps, like other apps, could inadvertently ship with sensitive information (e.g., credentials, private keys). The critical point here is that such information will be publicly available because the preferred delivery mechanism is through app stores. For example, as shown in Section 4.3, PickApp contains the password generation algorithm used to authenticate it to the endpoint. This means that anyone can download the app from the vendor's website, reverse-engineer it as we did, and discover the information required to interact with the endpoint on the factory floor.

We found cases of HMI apps being delivered directly by vendors rather than through app stores. In these cases, sideloading is the only installation option. Under this option, the vendor of an app instructs users on how to enable the "trust external apps" security setting, which allows the installation of any app, bypassing the end-to-end authentication and integrity checks backed by the app store infrastructure. Consequently, a repackaged version of an official app — to name one attack vector — would be accepted, with no way for average users to verify if the app is authentic or non-malicious.

Several of the apps that we downloaded from the Google Play Store contain unsafe, albeit not always directly exploitable, code patterns such as the use of external (shared) storage or the use of embedded web views with JavaScript fully enabled. These unsafe code patterns are easily found via automated code review. If unnecessary, such unsafe features should be disabled. But based on our findings, it seems that either the developers have ignored the output provided by automated code review tools or no such code review has been performed at all.

An attack that exploits a vulnerable mobile HMI is described in Section 4.3.

3.3.4 MES

In a smart manufacturing system, the MES serves the critical role of being the gateway between high-level manufacturing scheduling (e.g., ERP) and the manufacturing floor, where the goods are actually produced. The MES market is "closed" and oriented toward ad hoc solutions. For example, we were able to find only two open-source solutions,[52, 53] as opposed to many more commercial and enterprise products,[54] the most popular of which include General Electric's Predix Manufacturing Execution Systems, Honeywell Connected Plant, Rockwell Automation's FactoryTalk ProductionCentre MES, SAP Manufacturing Execution, and Wonderware MES.

Enterprise-level MESs are very expensive, and it is difficult to gain access to them. From a security research perspective, this is clearly a problem because of the importance of having access to a real, full-fledged system for security testing. Finding internet-facing MESs is highly unlikely, apart from cloud-based solutions and some random transient instances of Wonderware that we found exposed via Remote Desktop Protocol (possibly a honeypot or staging system).

Looking at an MES from a security standpoint, we can reasonably assume that the attacker is already within the network. The question we want to answer is about lateral movements. This does not mean that we assume that the attacker has access to the MES (otherwise, it would already be too late). For example, we consider an attacker that can access only the database of the MES and not the entire MES endpoint.

An attack involving data mangling on the MES is described in Section 4.4.

3.3.5 Complex, Programmable Manufacturing Machines

Complex programmable machines such as industrial robots execute their manufacturing tasks according to task programs, which are essentially scripts executed on the machine side (e.g., “move right,” “open pliers,” “move down,” “pick up piece”). Each machine vendor has its own domain-specific language for writing task programs, such as ABB’s Rapid, Comau’s PDL2, Fanuc’s Karel, Kawasaki’s AS, the Kuka Robot Language (KRL), Mitsubishi’s Melfa Basic, and Yaskawa’s Inform. These industrial robot programming languages (IRPLs) are all proprietary, and each of them has a unique set of features.

IRPLs can be very powerful because they allow programmers to write automation programs that also read-write data to or from the network or files, access the process memory, execute code downloaded dynamically from the network, and so on. One of the main use cases of such powerful features is the need for integration with middleware software, i.e., to let a robot talk to a vendor-neutral solution such as the Robot Operating System Industrial (ROS Industrial), which is the most popular solution (with many top industry brands being part of the ROS Industrial consortium[55]).

If used improperly and without the right security mindset in place, these powerful functionalities could be very dangerous.[56] First, if used without input validation (which is the most common case we discovered), these functionalities could introduce vulnerabilities. Second, because there is no privilege separation during execution, a program that performs simple machine movements is indistinguishable from a program that reads from the network, writes on a file, or executes that file (i.e., a dropper-like behavior), or from a program that scans the network to find targets, harvests and exfiltrates files on the manufacturing machine, or alters robot movements and other properties that affect the physical environment.

The attack vector can simply be a malicious task program, which will not be detected by conventional security scanners (similar to how PowerShell or JavaScript malware variants used to go undetected), or a trojanized task program with a dropper functionality, which will download the malicious payload and execute it when it is unexpected.

A technical description of how the vulnerable or malicious automation logic in a complex manufacturing machine can be exploited is in Section 4.5.

04 Case Study: Attacks

In this section, we describe how we tested the feasibility of some attacks under different attacker model assumptions. In certain cases, we assume that the attacker does not have direct access to the smart manufacturing (floor) network, while in others, we explain what the consequences would be if the attacker could access the network.

Figure 13 shows a high-level overview of the possible attacks included in our case study. We focus on the three entry points described in Section 3.2 (purposely leaving out the classic infected USB flash drive entry point à la Stuxnet,[57] which had been analyzed quite a few times in previous researches): the MES database (or the ERP system, alternatively), the engineering workstation (which is used to create and deploy automation logic), and a custom IIoT device (which can be a custom-developed embedded system). We describe two attacks — compromise through a malicious industrial add-in and trojanization of a custom IIoT device — through which an actor can gain access to the entry points indirectly. We then describe three attacks — exploitation of a vulnerable mobile HMI, data mangling on the MES, and use of the vulnerable or malicious automation logic in a complex manufacturing machine — that would allow lateral movement (e.g., on the HMI or the MES) or persistence (e.g., on the robot). The key details of these attacks — presented in this paper according to the depth of their penetration into the system, from entry point to the final target — and their corresponding defense approaches are listed in Table 2.

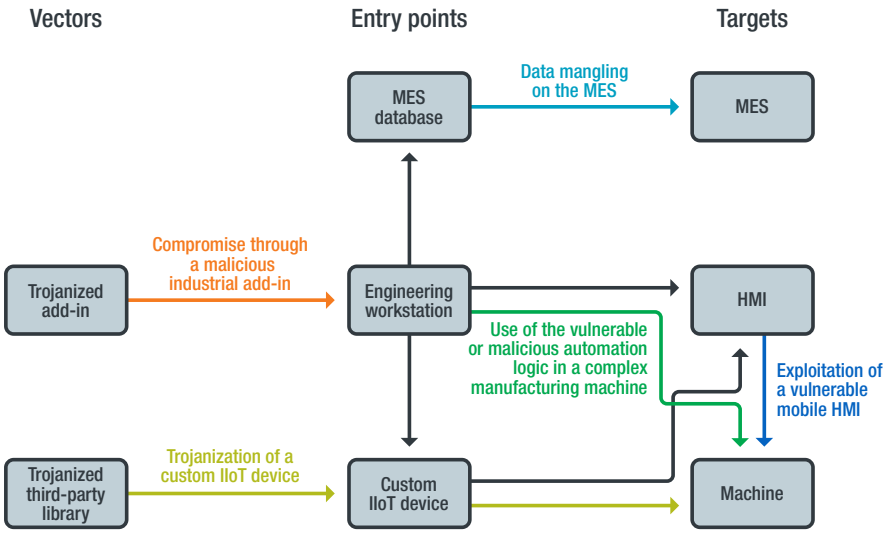


Figure 13. A high-level overview of the possible attacks included in our case study

Attack	Attacker is	Target is	Attack technique	Impact	Ease of attack	Ease of defense	Defense approach
Compromise through a malicious industrial add-in	Remote	Engineering or development workstation	Indirect: malicious software extension	Very high (full plant compromise)	Easy to medium	Medium (software extension vetting and isolation)	File scanning prevents malicious software extensions from landing on the engineering or development workstation. Details in Section 4.1.
Trojanization of a custom IIoT device	Remote	Custom IIoT device	Indirect: trojanized library or compromised repository	Very high (full plant compromise)	Medium	Medium (software supply chain security)	Visibility over the software supply chain validates every third-party component used in the development of IIoT software. Details in Section 4.2.
Exploitation of a vulnerable mobile HMI	On network	Mobile HMI	Credential harvesting	Medium	Medium	Easy (mobile app vetting)	Mobile app vetting is the minimum required to avoid vulnerable or malicious apps. Details in Section 4.3.
Data mangling on the MES	On network, database or ERP system	MES	Database data or network spoofing	High (altered product)	Medium	Easy (endpoint or network monitoring)	Prevention of database or ERP system compromise is key since all traffic from the database or ERP system to the MES is normally authorized. Countermeasures against Internet Protocol (IP) and Address Resolution Protocol (ARP) spoofing should also be deployed Details in Section 4.4.

Attack	Attacker is	Target is	Attack technique	Impact	Ease of attack	Ease of defense	Defense approach
Use of the vulnerable or malicious automation logic in a complex manufacturing machine	System integrator, or from compromise through a malicious add-in	Machine (e.g., robot)	Vulnerable or malicious automation logic	High (hijacked physical machine)	Very hard	Hard (custom program analysis engine)	Custom program analysis techniques validate each automation logic before deployment, at the system integrator level. Details in Section 4.5.

Table 2. Details of the attacks included in our case study and their corresponding defense recommendations

Overall, we envision the following multi-attack flows:

- **Attack: compromise through a malicious industrial add-in → Entry point: engineering workstation → Attack: use of the vulnerable or malicious automation logic in a complex manufacturing machine (robot)**
  - The engineering workstation is compromised via a malicious industrial add-in.
  - The malicious industrial add-in appends malicious automation logic code that will be deployed on the robot.
  - The malicious automation logic code:
    - Maps the network.
    - Exfiltrates files and network information from the robot host.
    - Implements a malicious server to support the other steps of the attack.
- **Attack: trojanization of a custom IIoT device (open-source library) → Entry point: custom IIoT device**
  - Either the development workstation or the repository of an open-source project library is compromised.
  - The library is altered to:
    - Report incorrect temperature readings so that the plant will stop due to safety rules.
    - Perform Address Resolution Protocol (ARP) spoofing as a noisier alternative to stop the plant.
  - The developer creates custom firmware to monitor the temperature readings and unknowingly includes the trojanized library.

- **Entry point: MES database → Attack: data mangling on the MES**
  - The attacker obtains access to the database of the MES.
  - The attacker reverse-engineers the structure of the database.
  - The attacker alters one number in one row of the database.

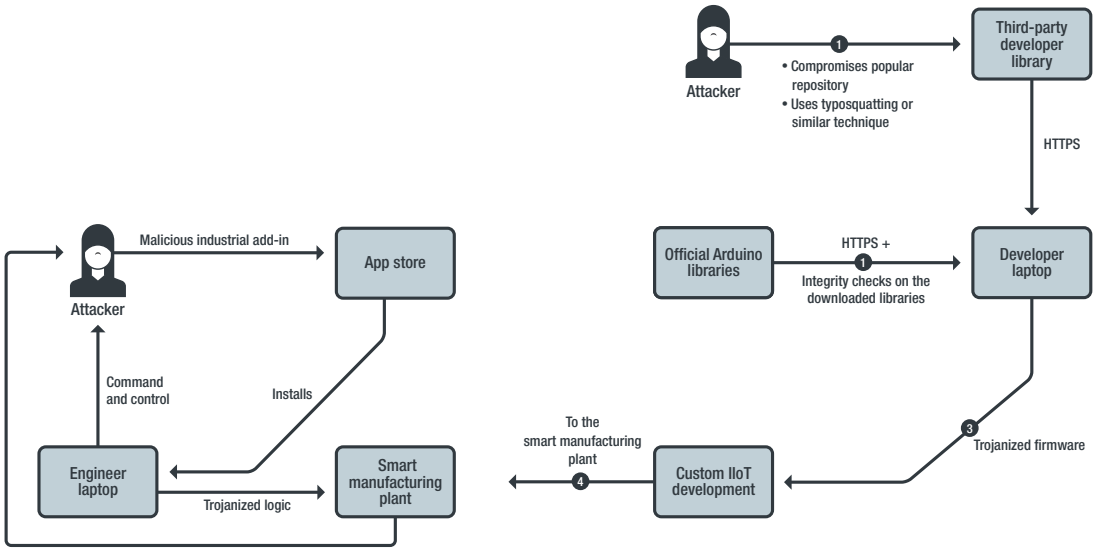


Figure 14. The overall multi-step attack scenarios that we envision given our findings described in the remainder of this section, including initial compromise using a malicious industrial add-in (left) and a "backdoored" third-party library (right)

4.1 Compromise Through a Malicious Industrial Add-in

In this section, we show, using tools and development environments that are specific to industrial automation, how an attacker can gain access to the target engineering workstation, steal secrets from it, and remotely trojanize every task program developed, in order to move laterally to the smart manufacturing system.

The attacker's end goal is to alter the production or remain persistent in the smart manufacturing plant, even if the engineering workstation is not directly connected to it (e.g., the engineering workstation could be a consultant's laptop, used to develop the automation logic that would be delivered to the customer).

Defense approach: Malicious industrial add-ins are essentially DLL files. Since DLL files do not represent a threat per se, the best defense approach is to use behavioral endpoint protection solutions, which can detect whether an executable is trying to harvest and exfiltrate files, among other actions.

Background concepts and a security analysis of industrial add-ins, which are relevant to the discussion of this attack, are in Sections 3.2.1 and 3.3.1.

4.1.1 Industrial Add-ins as Attack Vectors

In this section, we explain how a vulnerable development environment could be exploited by an attacker as a first, indirect step toward gaining access to the entire smart manufacturing system. We estimate that an attacker in this case would be able to infect about two distinct computers per day.

We found out that ABB's app store was affected by a file-upload bypass vulnerability, in which RobotStudio made non-approved add-ins immediately available. We reported this finding to ABB, which acknowledged and fixed the vulnerability. In addition, we verified that it is possible to create add-ins that collect data from the target engineering workstation and send it out over the public internet. We also verified that it is possible to create add-ins that append malicious code that will be delivered to the robot. (It should be noted that RobotStudio is used to develop, among other things, automation logic, specifically in the Rapid programming language.) Essentially, these add-ins have access to all of the system resources that are necessary to implement any functionality. As shown in Figure 15, the normal workflow is such that a developer could install add-ins from the app store via the web view (e.g., using a browser) or via the desktop view embedded in RobotStudio.

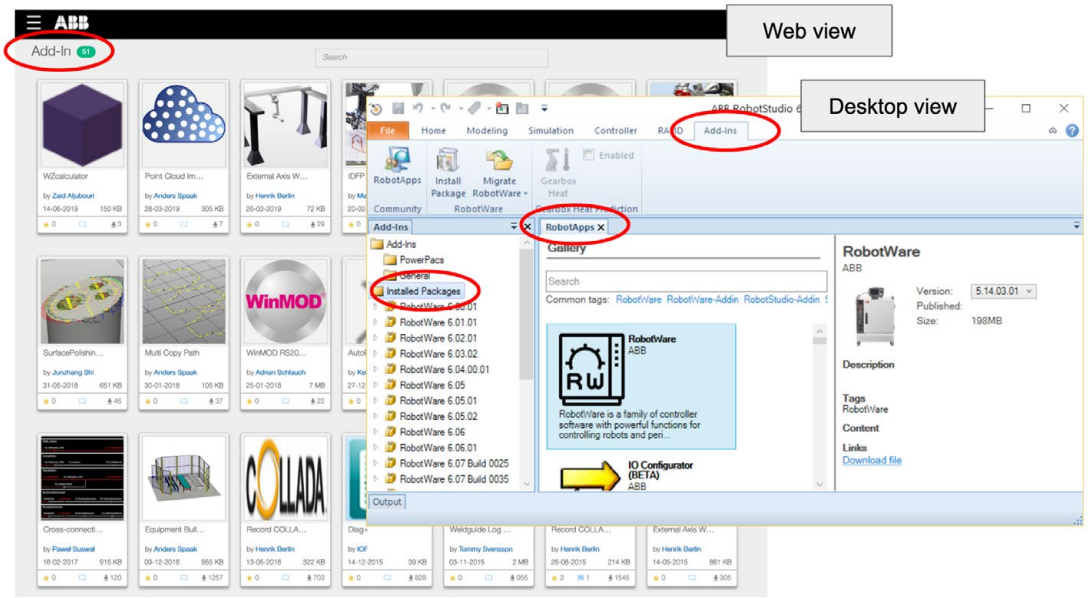


Figure 15. Add-ins can be downloaded from the web-facing app store and the app store client built into the desktop application (RobotStudio), the latter of which makes add-in installation seamless with just the click of a button.

Going into more detail, we discovered three issues (from high- to low-level) involving add-ins uploaded to the ABB app store.

4.1.1.1 Weak Vetting Process

Anyone can register and upload add-ins; there is no developer registration process as seen with mobile app stores. Moreover, there is no strict vetting of the uploaded code. In our case, it was possible for one of our research partners to successfully upload a (harmless) add-in that included the following note: "[…] prepared it and uploaded it to check whether this app store has any manual vetting procedure. […] to check whether someone would be able to uploading [sic] software, including non-benign software, via this app store." The add-in was readily accepted, and within 10 days it had been downloaded by 18 users.

4.1.1.2 Lack of Sandboxing

Although the uploaded add-in was harmless, we verified that, once downloaded, an add-in can do anything on the system, including network communication and file system harvesting, without any sandboxing or other restrictions. To this end, we created an offline add-in that performed these actions.

As shown in Figure 16, a test add-in that we created was able to recursively walk the C:\Users directory tree, collect full file paths from it, and make web requests, e.g., to send out harvested information to a remote endpoint.

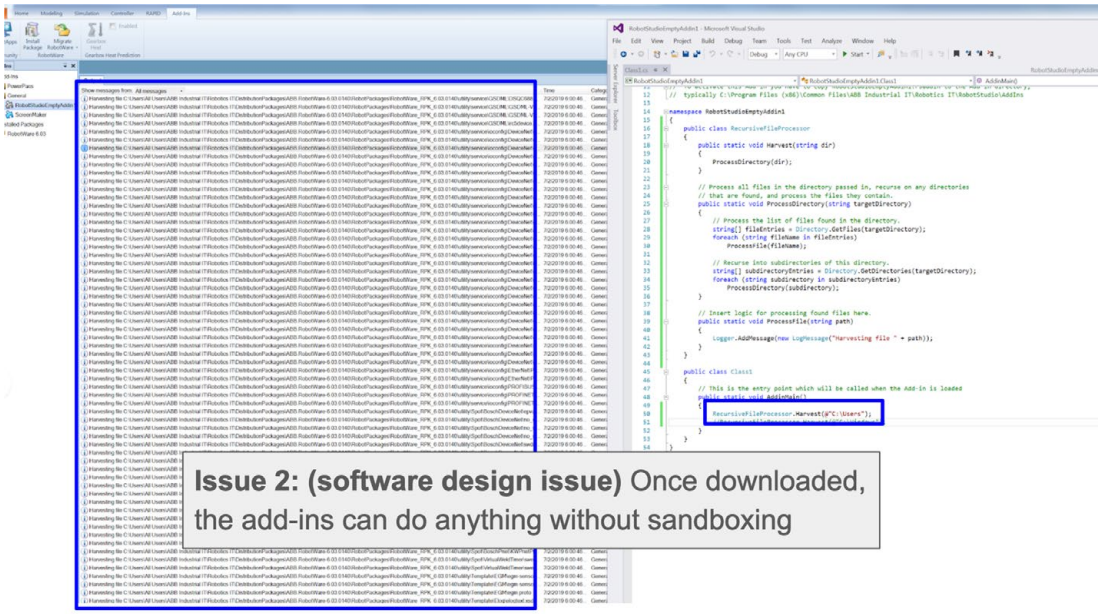


Figure 16. Once downloaded (and automatically installed), an add-in is not sandboxed and can perform actions on the system using the same privileges of the host app.

4.1.1.3 File-upload Vetting Bypass

While waiting for approval, the uploaded add-in was not showing up in the web view of the app store. But it was immediately available for download through the built-in desktop app store interface of RobotStudio, as shown in Figure 17.

ABB has fixed this server-side issue in response to our disclosure. A completely remote attacker could have indirectly infected the users of RobotStudio by uploading a malicious add-in that would bypass the vetting process.

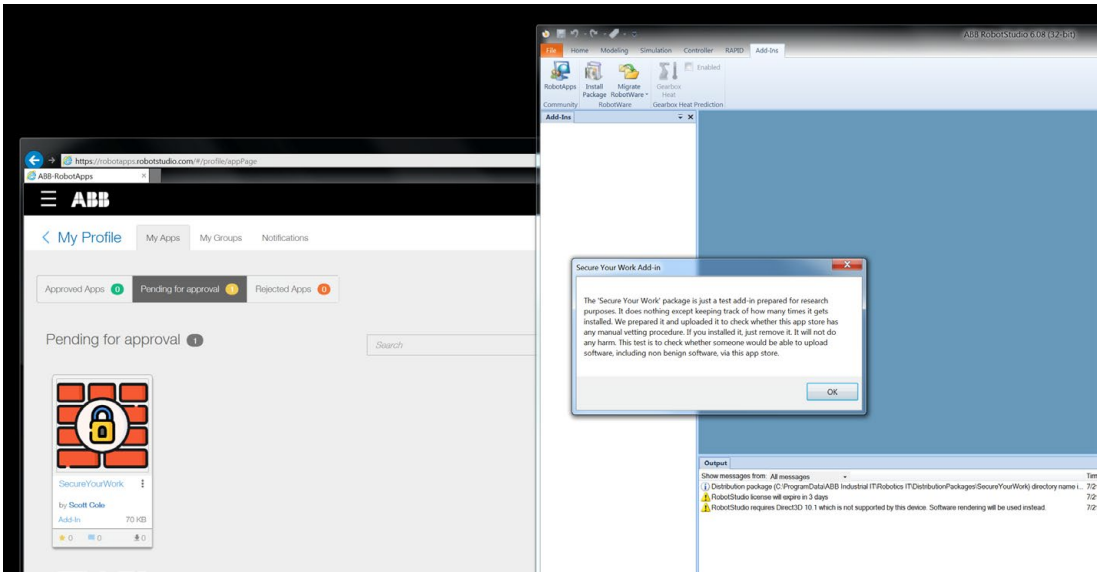


Figure 17. While still waiting for approval, the test add-in could already be downloaded and installed.

4.1.2 Digital Twin Compromise

We found and reported a slight variation of the aforementioned issues in the Kuka development ecosystem, specifically in the engineering and development software Kuka.Sim, which is used for both robots and computer numerical control (CNC) machines. An advisory for this vulnerability, which has been assigned the identifier CVE-2020-10635, has been published by the Industrial Control Systems Cyber Emergency Response Team (ICS-CERT).[58]

Like any OLP software, Kuka.Sim is used to design and test automation logic programs offline, before on-site testing. The version that we tested, Kuka.Sim Pro 3.1, has a feature called eCatalog, which allows developers to import externally developed, interactive 3D models in their simulations. In other words, eCatalog contains the digital twins of the industrial machines, which we consider a fairly advanced and forward-looking function. Internally, each of these digital twins, examples of which are shown in Figure 18, is made up of a 3D model combined with the KRL source code that defines its physical behavior. Any modification in the code will be reflected in the behavior of the digital twin.

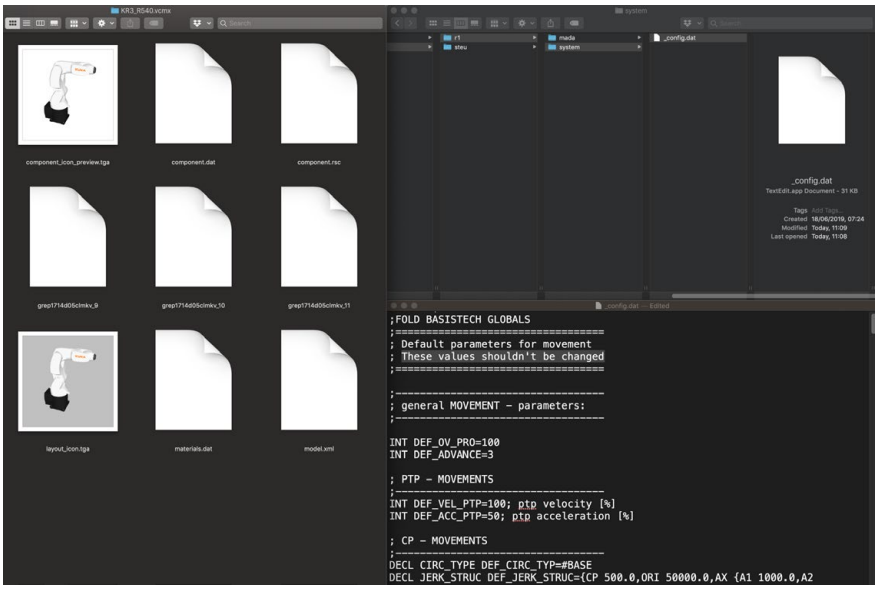


Figure 18. Like many other digital twins, each of Kuka's digital twins is made up of a 3D model along with simulation parameters, which "govern" its physical behavior when in use.

We found two security issues that, despite their simplicity, could have serious consequences if exploited.

4.1.2.1 Lack of Code Signing or Integrity

There are no application-level integrity checks on the data fetched from eCatalog, including the digital twins. The integrity of the digital twins is critical because they are used as the references for creating industrial automation tasks and workflows. They are simple ZIP files containing both code (written in the KRL) and 3D models (in 3D Studio format). Without any integrity checks, there is no way to verify whether the digital twins have been tampered with.

4.1.2.2 Unencrypted Network Transport

The network communication between the Kuka.Sim client and the remote eCatalog endpoint is over plain HTTP, not via HTTPS. Given the ubiquity of HTTPS thanks to the Let's Encrypt initiative,[59] we believe there are very few barriers to its adoption. With minimal time investment, anyone can set up an HTTPS endpoint, which provides an immediate gain in security.

The digital twins are fetched from the remote eCatalog servers as soon as the Kuka.Sim software boots. After having downloaded and installed a fresh copy of Kuka.Sim, we observed a lot of plain-text HTTP traffic to the visualcomponents.net host, which we later discovered was related to the eCatalog feature. The directory of the catalog is fetched via HTTP from "http://download.visualcomponents.net/elib/KUKA_Sim_3.1/components.xml". (There seems to be no HTTPS endpoint configured.)

Even if there were application-level integrity mechanisms to protect the files served from the directory (e.g., hashes or other checksums for Kuka.Sim to verify), they could not be considered secure because an MitM between the host computer and http://download.visualcomponents.net would be able to tamper with the XML code, including any integrity information. On top of that, even the individual digital twin files are fetched via HTTP, making it trivial for an MitM to tamper with them. In our responsible disclosure to Kuka, we suggested that they switch to HTTPS for any network communication from Kuka.Sim (or with any other software).

4.1.2.3 Tampering With Digital Twins

By exploiting the lack of integrity (at both the application and transport levels), an attacker — even a remote one — on the network can do several things. Since their integrity is not checked, an attacker can change the code, the 3D model, or both. Furthermore, the attacker can modify a digital twin by altering its visual 3D appearance, its reference system (coordinates), or both. An unaware programmer would then create a program for the machine based on the (altered) simulation parameters. The program would run smoothly when simulated on the machine that runs the altered digital twin.

Consequently, programmers may create projects based on tampered digital twins. When these projects are tested on real machines (e.g., robots, CNC machines), the effect is going to be unpredictable. The program will fail the on-site, preproduction tests, but it will be difficult to determine why. In addition, programmers will waste time figuring out why the simulation runs smoothly.

We verified this information with the aid of a field expert and reported the issue to Kuka.

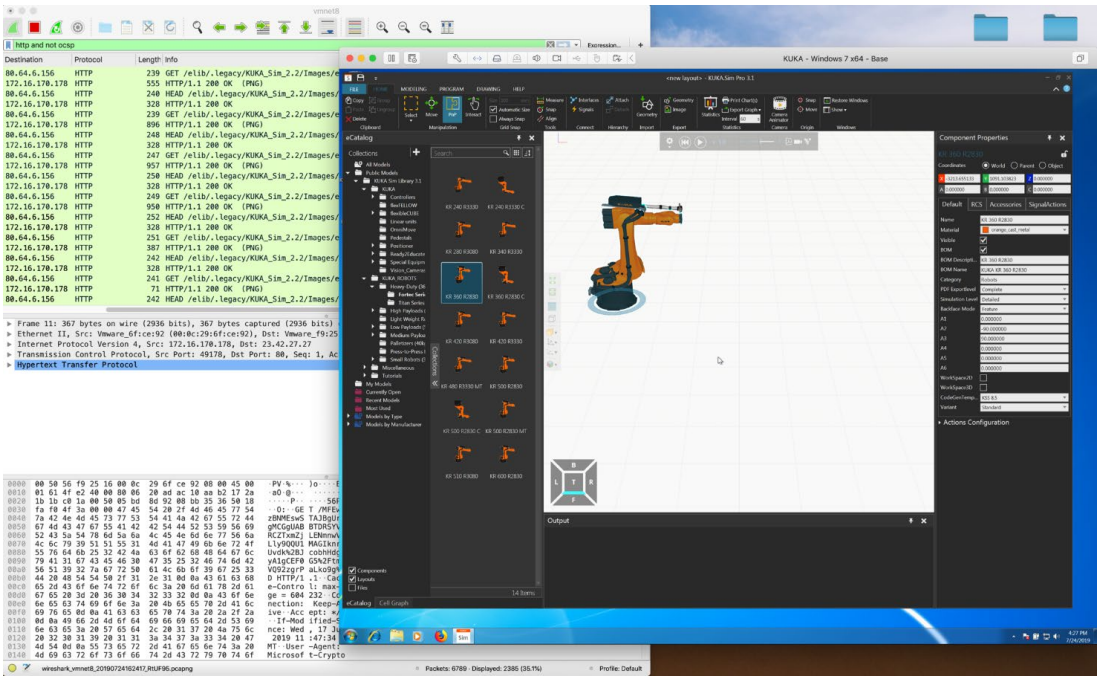


Figure 19. Kuka's eCatalog does not use any integrity protection at both the application and transport levels, making it possible for an attacker to modify the digital twins.

4.2 Trojanization of a Custom IIoT Device

In this section, we show how an attacker can compromise a smart manufacturing system through custom IIoT devices. Although the attack vector that we use is not specifically tied to smart manufacturing environments, this attack can certainly have an impact in such a setting.

We assume that the attacker wants to either cause malfunctions on the smart manufacturing system or facilitate other attacks on the network. In our attacker model, the attacker does not have direct access to the smart manufacturing system or its network. We assume that the attacker can access a developer's computer — possibly via the method described in the section on compromise through a malicious industrial add-in (Section 4.1) — either to alter an installed library or to compromise the original library repository (in the case of a more capable attacker).

Defense approach: The best approach is to have full visibility over the software supply chain, including the third-party components used (internally) by developers to build custom firmware for IIoT devices. This means that whenever a library is included in a software project — and many open-source libraries are used nowadays — it needs to be considered as untrusted, in the sense that a full code review needs to be performed whenever it changes.

Background concepts and a security analysis of custom IIoT devices, which are relevant to the discussion of this attack, are in Sections 3.2.2 and 3.3.2.

4.2.1 Security Risks of Modern (Industrial) IoT Development Practices

Arduino provides its own official integrated development environment (IDE) along with an online catalog of about 80 official libraries,[60] which are vetted and considered stable. The end-to-end integrity of these libraries is between the Arduino servers and the developers' computers that download them. While using the Arduino IDE, developers are assured that the libraries that they download are authentic, and their integrity is guaranteed by a cryptographic hash. From there, however, whatever happens on a developer's computer is outside Arduino's control. There is no way to guarantee that the code that will end up being loaded and executed on the IoT node has not been tampered with. Another risk, which is mitigated by the vetting at the source, is that the original library developer could have included a malicious functionality or a vulnerability in the library, e.g., the original developer's repository could have been compromised.[61, 62, 63, 64] For this, Arduino is the trusted third party because it (periodically) checks all of the official libraries in the official catalog.

Modern embedded development ecosystems are often extremely large and complex, and are filled with a variety of resources, tutorials, and libraries that are casually uploaded to code repositories and other, similar places. Developers are inclined to take libraries apart, modify them, and then re-upload them somewhere else, copying code from and onto community forums such as Stack Overflow. Moreover, advanced developers do not rely on the Arduino IDE; they are used to incorporating libraries coming from unofficial sources. The most representative example is PlatformIO, a fairly advanced development environment that also offers a catalog of more than 7,000 libraries.[65] On PlatformIO, anyone can register a library for other developers to download. There is no end-to-end integrity mechanism apart from the fact that the libraries must be downloaded via HTTPS. In other words, there is no way to detect whether the source repository that hosts the library has been compromised or even whether it still contains the original code written by the developer, as with what happens in the official Arduino libraries.

Furthermore, the Arduino hardware abstraction layer is compatible with several target boards — which is its main goal. For example, even if the Siemens Simatic IoT2040 is meant to run Yocto Linux and its official Siemens software distribution, any custom firmware will run seamlessly on it, without any hardware modification. There are several tutorials that clearly explain how to do this. Aside from the Siemens Simatic IoT2040, there are many industrial devices that are compatible with Arduino, as noted in Section 3.2.2.

4.2.2 Attack Demonstration

To avoid calling out a specific brand or product, which is not what we aim to do, we demonstrate the attack using generic Arduino-compatible devices from an unknown brand, like the one shown in Figure 20. Indeed, the brand is not relevant in this case because the issue is unrelated to the vendor but rather lies in the (open-source) software supply chain.

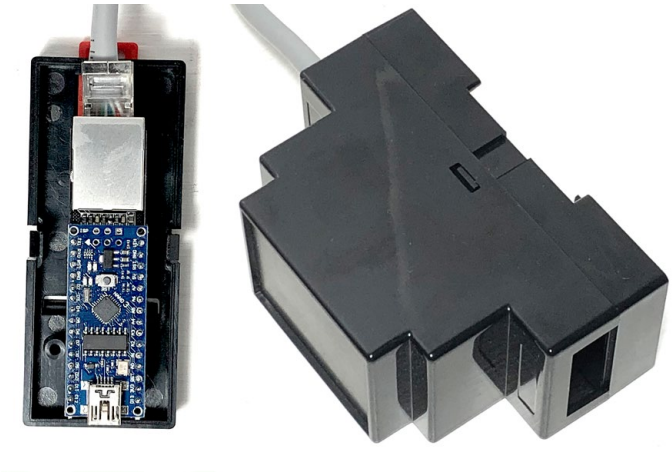


Figure 20. The custom IIoT device we used to demonstrate the attack

Given the above premise, we show how an attacker could trojanize firmware via a software supply chain attack targeting one of the libraries used by a developer. We assume that an IIoT developer wants to create a monitoring node that collects temperature readings, e.g., for predictive maintenance or anomaly detection. This is a very common scenario these days: multiple sensors attached to various key points in the production plant that detect signs of anomalous behavior (e.g., a machine that may be faulty or may be working beyond its physical limits). In Industry 4.0 Lab, the monitoring network is on a separate, air-gapped network (although this may not always be the case), and is used to send monitoring data to a cloud service, which activates an alarm (with a loud siren) every time the measured value is out of safety range. Figure 21 shows the main loop running on the custom IIoT device and the readings that are reported over time.

```
void loop() {
#ifndef SOLO
  if (eth_on)
    ether.packetLoop(ether.packetReceive());
#endif

  long now = millis();
  if (now - lastMsgTime > SAMPLING_DELAY) {
    lastMsgTime = now;

    // Reading BME280 sensor data
    bme.takeForcedMeasurement(); // has no effect in normal mode
    temperature = bme.readTemperature();

    if (isnan(humidity) || isnan(temperature)) {
      Serial.println("BME280 reading issues");
      return;
    }

    char str[6];
    dtostrf(temperature, 4, 2, str);
    Serial.println(str);

#ifndef SOLO
    if (eth_on) // Send temperature readings
      ether.sendUdp(str, sizeof(str), srcPort, srv, dstPort);
#endif
  }
}
```

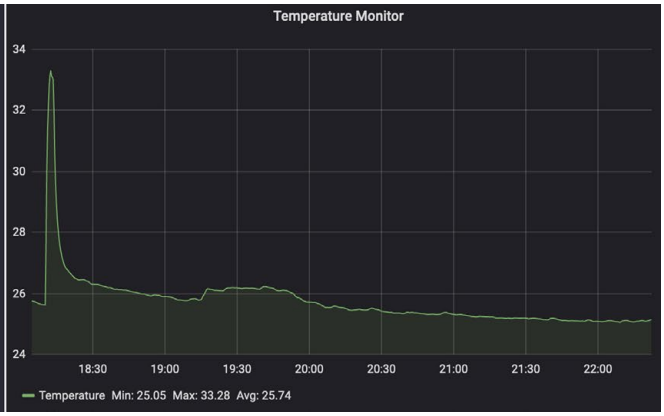


Figure 21. A timeline of the temperature readings

Because of the trojanized library, the developer is not aware that the firmware will report incorrect temperature readings after some time, as shown in Figure 22. From then on, an anomaly will be triggered, the alarm will go off, and any further reaction procedure will be engaged.

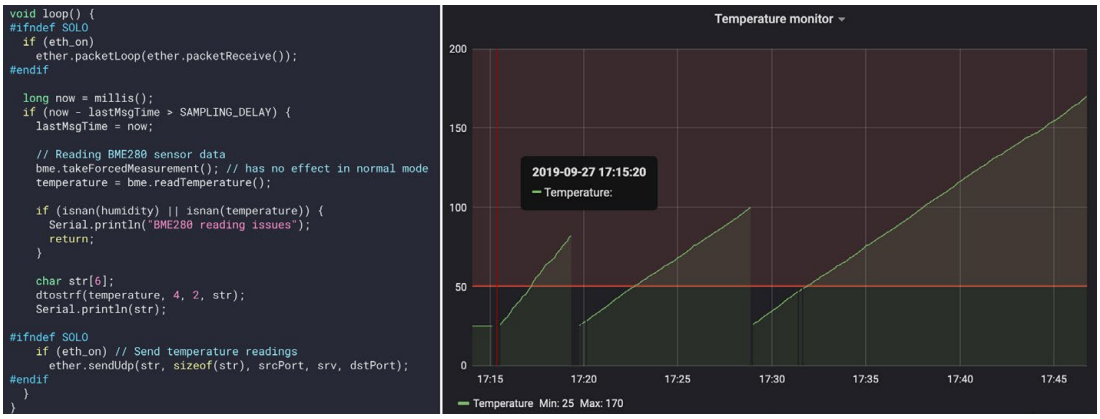


Figure 22. A timeline of the altered temperature readings

As an alternative strategy, the attacker could use the trojanized library to cause a denial of service (DoS) in the smart manufacturing plant by including less subtle malicious functionalities. As shown in Figure 23, an ARP spoofing loop could be included by the attacker. While the code that the developer wrote was the same as the one above, the library had been compromised by an attacker to launch an ARP spoofing attack at random intervals, which would disrupt network communication.

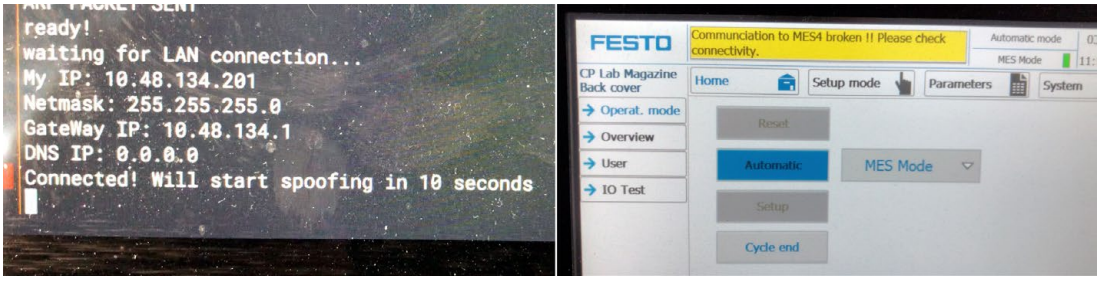


Figure 23. ARP spoofing caused by a trojanized library included in the custom IIoT device responsible for monitoring the temperature

4.3 Exploitation of a Vulnerable Mobile HMI

In this section, we explain how an attacker can obtain control of a connected machine by exploiting a vulnerable mobile HMI. For our attacker model, we assume that the attacker has access only to the (wireless) network that connects the mobile HMI and the connected machine.

Defense approach: Detecting vulnerable or malicious mobile apps can help prevent the root cause that an attacker might exploit. In the specific case of our case study, we found that the developers implemented a weak authentication scheme, which stored secrets right in the code. We do not think that this case would be detected by streamlined code analyzers. Thus, in this specific case, the best defense approach would be manual code review of the mobile app that implements the HMI.

Background concepts and a security analysis of HMIs, which are relevant to the discussion of this attack, are in Section 3.3.3.

4.3.1 Controlling Connected Machines via Mobile HMIs

We found one public mobile HMI app leaking sensitive authentication information, which would allow an attacker to easily reverse-engineer the credentials stored in it, and then reuse the same credentials to authenticate and send movement commands to a Comau industrial robotics controller. We responsibly disclosed this case to Comau through the ZDI; the relevant vulnerabilities have been assigned the identifiers CVE-2020-10998 and CVE-2020-10999. In response to our disclosure, Comau indicated that it would be blocking downloads of the affected app and making a new, updated version of the app available. As of the publication of this paper, Comau said that it did not know when a new version of the app would be available, or if a new version would be available at all.

Like many modern vendors, Comau offers a mobile HMI in the form of an app. In this particular case, Comau's PickApp can be used to interact with the robot as if the operator were using a traditional robot teach pendant, which is normally a wired, dedicated device. As depicted in Figure 24, the UI has control soft buttons for interacting directly with the physical robot. We are seeing a trend, beyond the robotics domain, of functionalities being implemented in apps or on general-purpose devices like tablets.

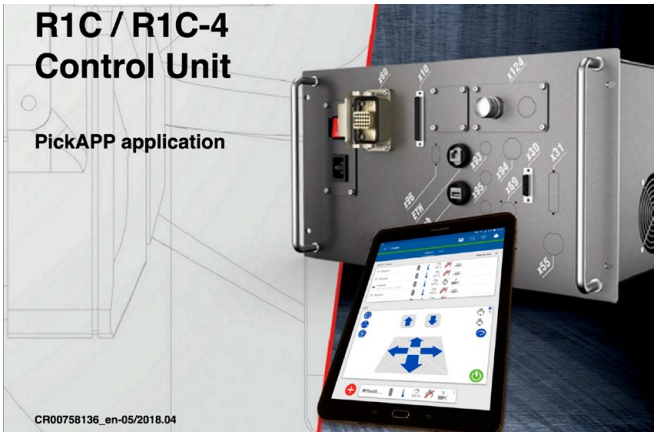


Figure 24. Comau's PickApp implements some of the functionalities of a robot teach pendant.

4.3.2 Credential Leaks

PickApp is free, so anyone can download it. While this is not an issue in itself, we managed to decompile and reverse-engineer the application package, where we found out the supposedly secret algorithm used to compute the password needed to connect to the robot (based on the calendar date and time), as shown in Figure 25.

```
public EDPValue login(String paramString1, String paramString2, MessageParameters paramMessageParameters)
{
  MessageObject localMessageObject = MessageObject.MessageObjectFactory(paramMessageParameters, this, EDPvoid.EDPvoidFactory()
  localMessageObject.m_TxStream.edp_encode_int(513);
  localMessageObject.m_TxStream.edp_encode_int(4);
  localMessageObject.m_TxStream.edp_encode_string(new EDPstr(paramString2));
  localMessageObject.m_TxStream.edp_encode_int(513);
  localMessageObject.m_TxStream.edp_encode_int(4);
  localMessageObject.m_TxStream.edp_encode_string(new EDPstr(paramString1));
  localMessageObject.m_TxStream.edp_encode_int(540);
  localMessageObject.m_TxStream.edp_encode_int(531);
  return transmit(localMessageObject);
}
```

Figure 25. From the source code obtained via decompilation, we found out the algorithm used to compute the password to connect to the robot.

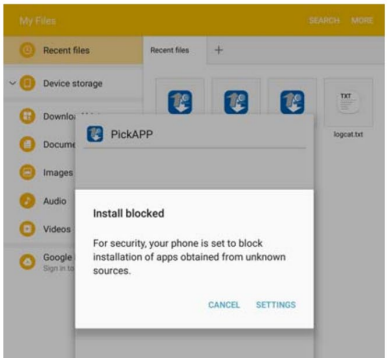
An attacker who has gained network-level access to a smart manufacturing plant that uses this line of robots, with PickApp enabled, can interact with the robot directly because they now know the credentials.

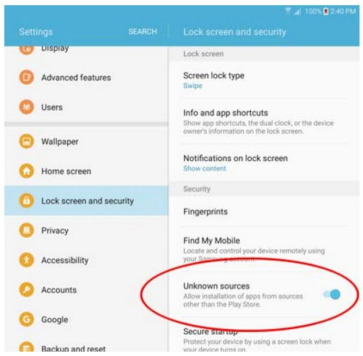
From a remediation viewpoint, the vendor needs to implement proper authentication protocols. This shows that a security upgrade is due for both the controller and application sides. Moreover, users need to create, deploy, and use proper credentials.

4.3.3 Application Sideloading

As shown in Figure 26, PickApp's manual instructs users on how to disable a security feature on their tablet or phone, which otherwise would prevent the app from running. It explains how to enable sideloading to allow the installation of applications outside official app stores, i.e., to allow the installation of PickApp, which is not officially distributed. This alone is an issue because users have no way to verify the authenticity of the app being installed. In addition, the manual neither warns users about this security risks nor explicitly states that users should re-enable the security feature once they are done with the installation.

a.1 the **Unknown sources** flag is NOT enabled in the tablet settings.





a.3 enable the **Unknown sources** flag (as shown in the figure above)

Figure 26. PickApp's manual explains in detail (including screenshots) how to enable sideloading of unofficially distributed apps like PickApp, without explaining the security consequences and advising users to disable it once the installation is finished.

To remediate this issue, the vendor should either distribute the app through the Google Play Store if the company policy allows for the use of internet connections (e.g., limited to the app store IP range) or create an internal "app store" with valid certificates if not.

4.3.4 Impact on Safety

Merely using software machine interfaces carries safety risks because there is no direct mechanical or electrical connection from the command-issuing device (e.g., the HMI) and the final actuator (e.g., the motor). The PickApp case, which is no exception, allows us to comment on this problem. The app substitutes the hardware teach pendant but is unable to fulfill the emergency stop time requirements (which are subject to standards and regulations). This is because a software failure on the mobile side may prevent the operator from issuing timely emergency stop commands.

As seen in the manual excerpt shown in Figure 27, the use of this mobile HMI requires the installation of a hardware bypass circuit, which transfers the emergency stop commands to the cell (e.g., robot) level. This assumes that the operator must stay close to the robot to issue an emergency stop command using the usual hardware red button.

Since in the ***PickAPP*** application scenario (refer to par. 2.2 Basic System on page 20) a mobile device is meant to substitute the Teach Pendant on Pick and Place processes programming and configuration, the related security issues are transferred at the cell level, due to the fact that commercial tablets do not contain neither the Emergency Stop button nor the Enabling Device.

For the described above reasons, in order to guarantee the operator's safety, robot manual movements (JOG), motors power on, START/STOP cycle commands, may come from ***the tablet*** but will take effect only if the cell gates are CLOSED; furthermore, the Emergency Stop button must be available at cell level.

For further details about the safety issues, refer to Chap.6 - Virtual TP5 Teach Pendant in R1C - R1C-4 Control Unit Use manual.

3.2 Bypass circuit

Figure 27. The PickAPP manual describes the impact on safety when software teach pendants such as PickApp are used.

4.4 Data Mangling on the MES

In this section, we show how an attacker can affect the proper operation of the MES. We assume that the attacker wants to cause malfunctions, damage the produced goods, or produce defective products. We assume that the attacker is not on the smart manufacturing system network and cannot access it, but can write one field in the database of the MES, which may or may not be on the same network.

Defense approach: The best strategy to avoid this attack is to prevent the database or ERP system from getting compromised because all traffic from the database or ERP system to the MES is normally authorized. Thus, if an attacker manages to compromise any trusted machine that can legitimately send data to the MES, it is already too late.

If the attacker is already on the network and if the MES accepts commands from the network, the attacker may try to spoof them. In this case, the recommendation is to deploy countermeasures against IP and ARP spoofing.

Background concepts and a security analysis of MESs, which are relevant to the discussion of this attack, are in Sections 2.1.1, 3.2.3, and 3.3.4.

While this attack is not tightly coupled with the Festo MES4 that we analyzed, it can affect any MES that does not implement data integrity checks on the database records. The MES we analyzed is running Festo MES4 1.1.0.9. This software is used to create a work plan and distribute it in the form of operations to the manufacturing line. A work plan specifies the operations that need to be executed from the line to process a piece and to create a final product. Each operation is executed by one distinct station, e.g., a drill that drills a hole through the piece under process.

The MES uses an internal database (Microsoft Access) to store the work plans that are created by the operator. When a work plan is executed, the MES uses this database to translate the work plan's operations in a series of parameter values for the different stations. For example, the operation "right drill" corresponds to Parameter Value No. 2 (table tblOperationParameter) and is used in communication with Station No. 3 (DRILL-CPS in tblResource).

We verified empirically that the MES neither authenticates the database nor contains a way of validating each of the records. This allows an attacker to arbitrarily tamper with the database and conduct two practical attacks. One involves the introduction of an error in the production. In this scenario, the attacker changes the value of the parameter associated with an operation, e.g., by changing No. 2 ("right drill") with No. 1 ("left drill"). As a result, every time the operator creates a "right drill" operation, this gets substituted by a "left drill" operation. Clearly, this attack would be even more effective if the attacker had the ability to alter the Q&A process, e.g., by tampering with the results of the camera check in order to return "OK" in the presence of defects.

The other practical attack involves blocking the production. As previously mentioned, each station is configured with a set of preconfigured available operations (e.g., "drill right" or "drill left"), which are configured in the PLC of the station. As a consequence, the station expects to receive one of these two values from the MES. We verified that it is possible to cause a DoS if an attacker introduces an out-of-bound value in the parameter values of one of these operations. For example, an attacker may replace Value No. 2 with Value No. 5. In this case, every time the operator creates a work plan with a "right drill" operation, the drilling station will trigger an error and block the production.

The attacks can be conducted either manually, by updating parameter values in the database (e.g., via an SQL update), or automatically, via malware.

A high-level demonstration of the attack sequence is shown in Figures 28 to 32.

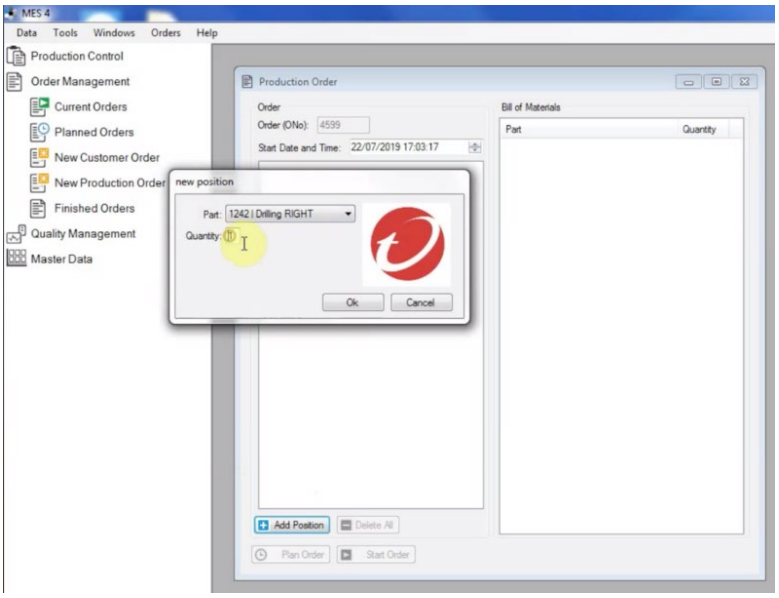


Figure 28. Data mangling on the MES, Step 1: A normal order is inserted via the MES and the production starts.

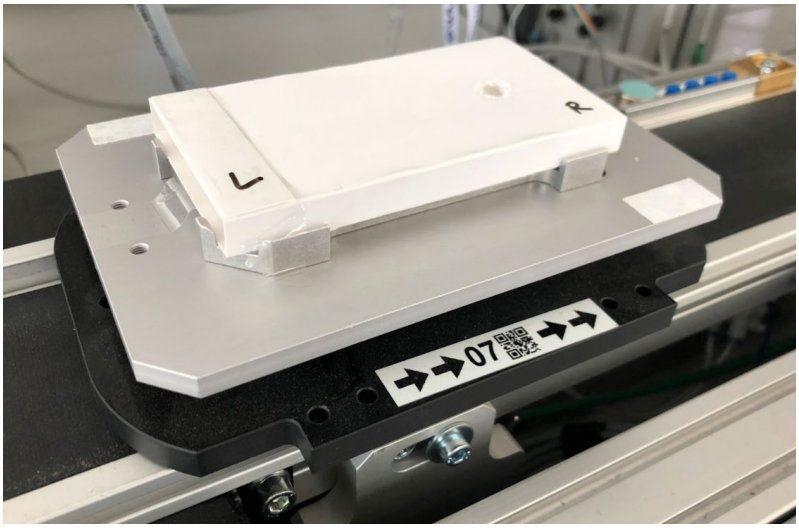


Figure 29. Data mangling on the MES, Step 2: The raw product (white box) enters the production line for drilling on the right.

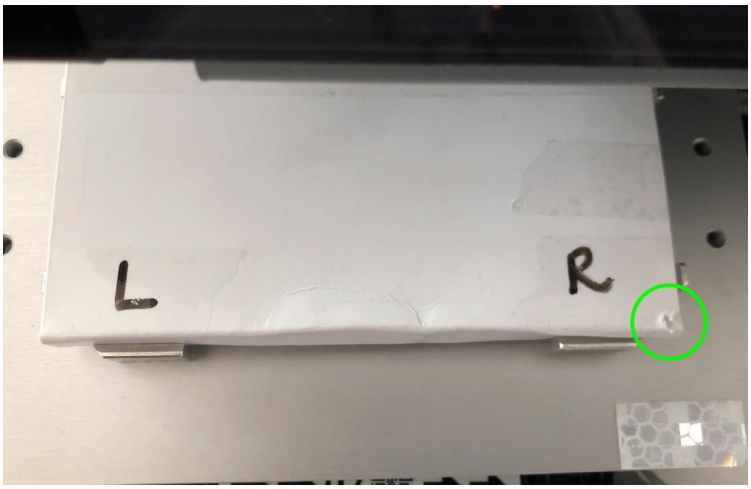


Figure 30. Data mangling on the MES, Step 3: The drill drills a hole in the correct position.

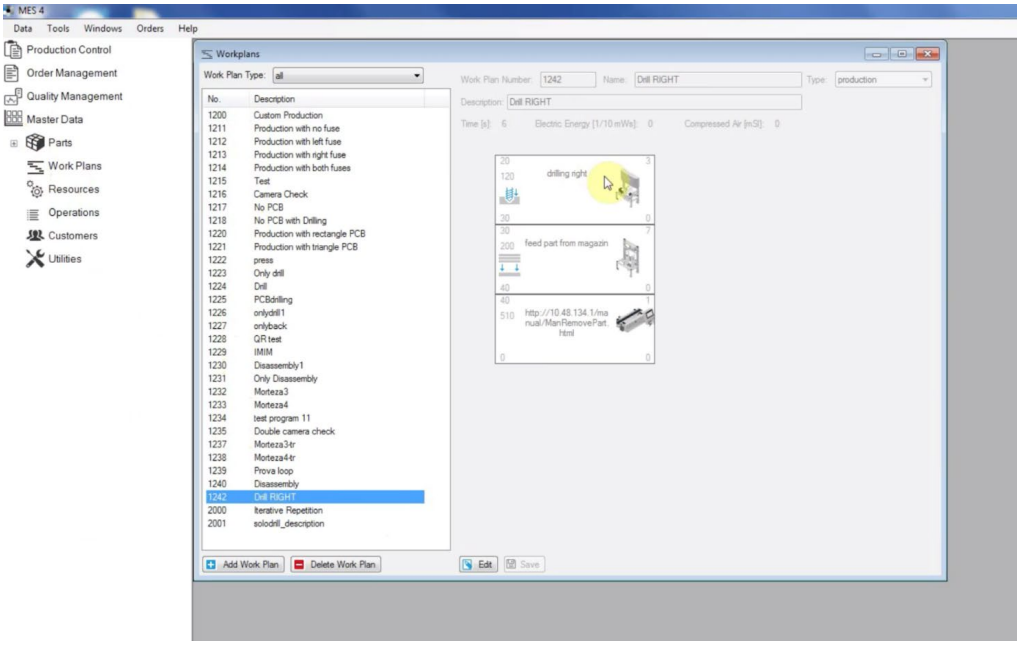


Figure 31. Data mangling on the MES, Step 4: The same order is restarted, again with a “drill right” instruction.

Figure 32. Data mangling on the MES, Step 5: The drill unexpectedly drills on the left.

4.5 Use of the Vulnerable or Malicious Automation Logic in a Complex Manufacturing Machine

Using the programming environments from major industrial robot vendors, we show that it is possible to create vulnerable or malicious machine automation logic. We also show how we found real-world cases of vulnerable task programs, one of which has since been removed by the vendor upon our responsible disclosure. In our attacker model, we assume that either the attacker is on the network and cannot access the target machine (e.g., robot) or the attacker is already on a target machine (e.g., robot) and wants to remain persistent while loading additional control logic, mapping the network, and exfiltrating sensitive information from it.

Defense approach: This situation requires custom program analysis techniques to validate each automation logic before deployment at the system integrator level. Unfortunately, network and endpoint monitoring are not enough for a couple of reasons. First, there are legitimate reasons that a robot, for instance, must receive data from a machine, and blocking that traffic will result in the disruption of the machine's functions. Second, automation logic is not compiled in common executable formats such as the Portable Executable (PE) format and the Executable and Linkable Format (ELF), neither of which is written in general-purpose languages such as C, for which scanners – which can point out vulnerabilities or signs of malicious behavior — are readily available.

Trend Micro has developed a technology with specific countermeasures for this attack. Certain portions of this technology are patent pending.

Background concepts and a security analysis of industrial robots and IRPLs, which are relevant to the discussion of this attack, are in Section 3.3.5.

Task programs for complex manufacturing machines are among the industry's best-kept secrets since they are essentially the digital twins of manufactured goods and, as such, carry a lot of value. For this reason, we were not expecting to find any public or open-source task program to analyze for vulnerable or malicious code patterns. However, by using a mixture of web search techniques (e.g., "site: github.com") and GitHub's search feature, we were able to crawl GitHub and similar code-hosting sites and collect nearly 2,000 task program files covering major industrial robot vendors. Since we knew the file extensions used by each vendor and the characteristic language keywords, we were able to narrow down the search and find several repositories containing real code. Despite most of the programs' being clearly used for educational purposes, we found a recurring and interesting use case: integration programs. These programs allow the connection of an industrial machine to on-premise services, either by exposing a network service on the machine's side or by acting as a client to an existing network service.

We extended this finding further by using our knowledge of the domain-specific programming language of each vendor to show three patterns for vulnerable programs and one for malicious programs. The most striking aspect is that, while it is possible to introduce input validation or logic vulnerabilities, the vendors' languages are rather limited in their string-processing functions. Therefore, it is hard to implement input validation routines. Moreover, and most importantly, the lack of cryptographic primitives makes it practically impossible to implement proper authentication or integrity checks on the data coming from the network.

Universal Robots is very peculiar and different from all of the other vendors because it allows the writing of automation logic in a general-purpose programming language (Java and Python being the most representative examples). In other words, Universal Robots is not limited to using its own IRPL. On one hand, this may sound very dangerous because it gives complete power to a malicious developer to write advanced malware with direct access to all of the hardware resources. On the other hand, it is a security advantage: All of the best practices and state-of-the-art code analysis tools (e.g., for Java and Python) can be used to find vulnerable code patterns, and general-purpose programming languages are powerful enough to allow the implementation of strong security measures such as authentication and encryption.

The mere presence of the named program language features does not represent a security issue. Only its unsafe use can create venues for vulnerabilities, the degree of exploitation of which depends on several conditions, as explained in the remainder of this section.

Figure 33 summarizes the broad cases of unsafe automation logic. On one hand, developers can introduce vulnerabilities by using unsafe programming patterns. On the other hand, malicious actors can purposely write programs that abuse specific functionalities. There is an interplay between the two cases, which can lead to malware that loads remote external code, enumerates the network (e.g., to find further targets), exfiltrates secrets, modifies configuration, or causes general damage.

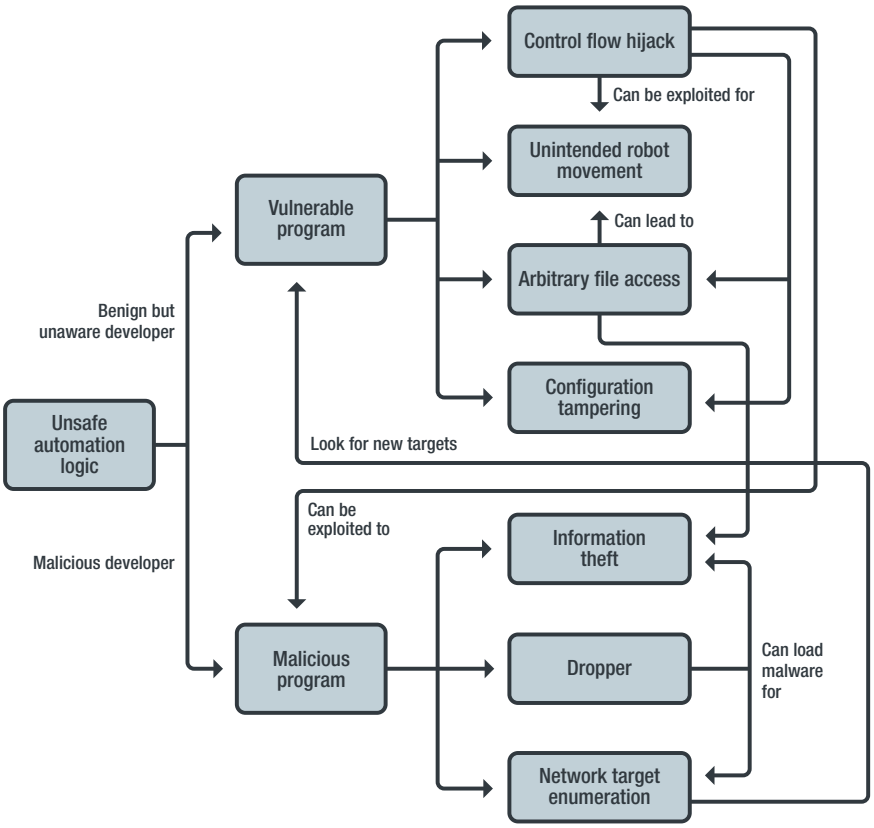


Figure 33. A summary of the broad cases of unsafe automation logic

4.5.1 Arbitrary File Access or Configuration Tampering

We found and reported a real case of a vulnerable web or file server implemented in ABB's Rapid language that was meant to run on an industrial robot.[66] As summarized in Figure 34, the file server was affected by a classic input validation vulnerability, which would have allowed an attacker to pass any string as the file path — including directory separators (e.g., "../"). This made directory traversal possible, allowing an attacker access to any files on the machine's controller computer. By checking the features of the other vendors' programming languages, we verified that a programmer can make similar mistakes on other platforms, so the issue is not limited to this case.

Also by looking at the features of the other programming languages, we concluded that variations of this vulnerable code pattern exists. For example, if there is an "unsanitized" path from an inbound communication primitive (e.g., network socket, field bus, serial) and a configuration-handling or file-writing function, then an attacker could exploit it to tamper with sensitive configuration files. This could result in instances like unexpected behavior from the controlled machine or even in the entire programming logic's being replaced by arbitrary files at the attacker's will.

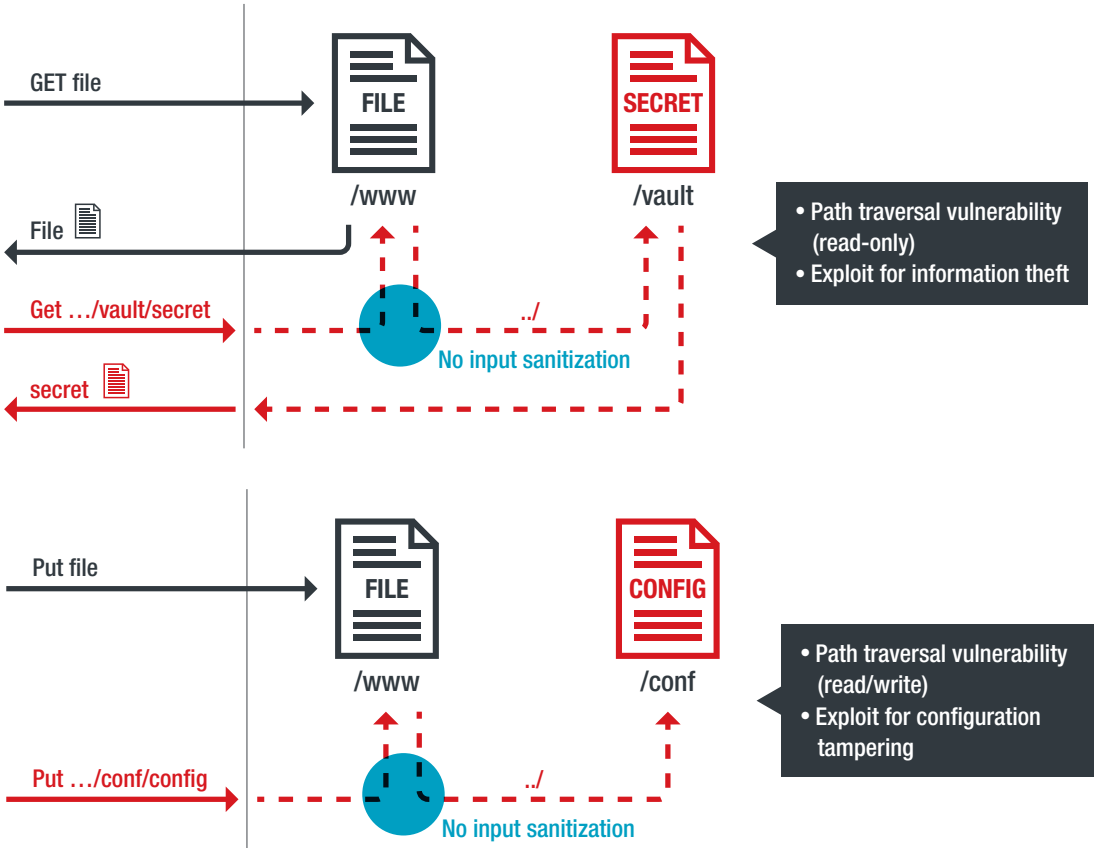


Figure 34. Vulnerable code pattern: from unsanitized (e.g., file, network, serial) data to read/write file access

4.5.2 Arbitrary, Unintended Movement Commands

A vulnerable code pattern emerged after we found that in all of the cases that we examined, there was no authentication or input validation on the movement coordinates. This means that an attacker who can send data on the network is also able to issue any movement command to the machine since there is no granular control on the coordinates sent to the robot. While seemingly straightforward at first, this vulnerability is difficult to mitigate in the context of a smart manufacturing environment, where every endpoint is trusted by default. A firewall will not help because even if machines are whitelisted, there is no way to block or control unexpected movement commands from those trusted machines. The only solution is to authenticate all packets. Implementing this at the source-code level is challenging because cryptographic support is either absent or very limited.

The only effective mitigation is to implement an authenticated and encrypted connection, such as a virtual private network (VPN) connection, between the robot and each of the endpoints that need to communicate with it. However, this can prove impractical, and even if it is implemented, the lack of input validation on the movement coordinates will still need to be taken care of at the source-code level (e.g., by checking values against boundaries).

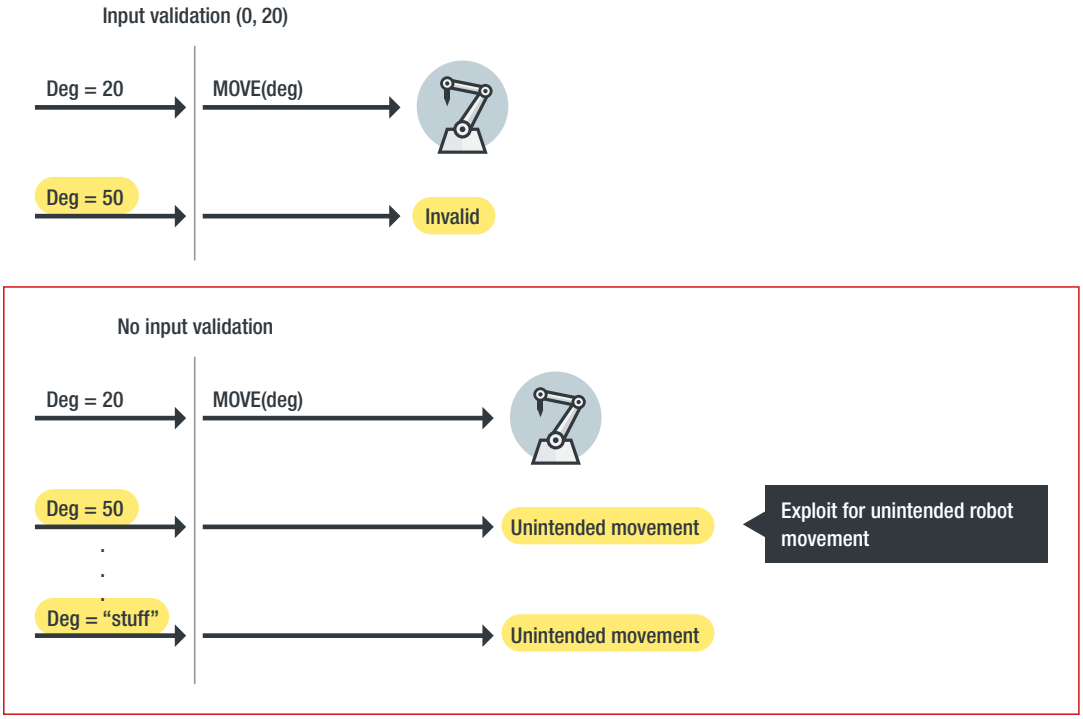


Figure 35. Vulnerable code pattern: From unsanitized (e.g., file, network, serial) data to movement commands

4.5.3 Control Flow Hijack

Some of the programming languages that we analyzed have a very powerful feature, called late binding or call by name, that is normally available in general-purpose, high-level programming languages such as C, C++, Java, and Python. Basically, this feature allows a programmer to write code that programmatically calls another routine by its name. Thus, instead of statically writing, say, `function_reset_coordinates()`, a programmer can write `callbyname(func_name)`, where `func_name` can be the `function_reset_coordinates` string value (or any other dynamic value).

If there is no input validation on the `func_name` variable in question, and if `func_name` is controllable by the attacker (e.g., because it comes from or is influenced by inbound network data), then the program is vulnerable. Not only did we find out that it is possible to implement programs that have this vulnerability, but we also found an instance of this vulnerability in an open-source, educational program.

By exploiting this vulnerable code pattern, an attacker can change the original automation logic behavior completely (e.g., by calling other code already present on the machine or by creating DoS loops).

The next section describes how the call-by-name feature is essential in creating advanced dropper-like malware whose operation differs from the loading of the actual malicious payload.

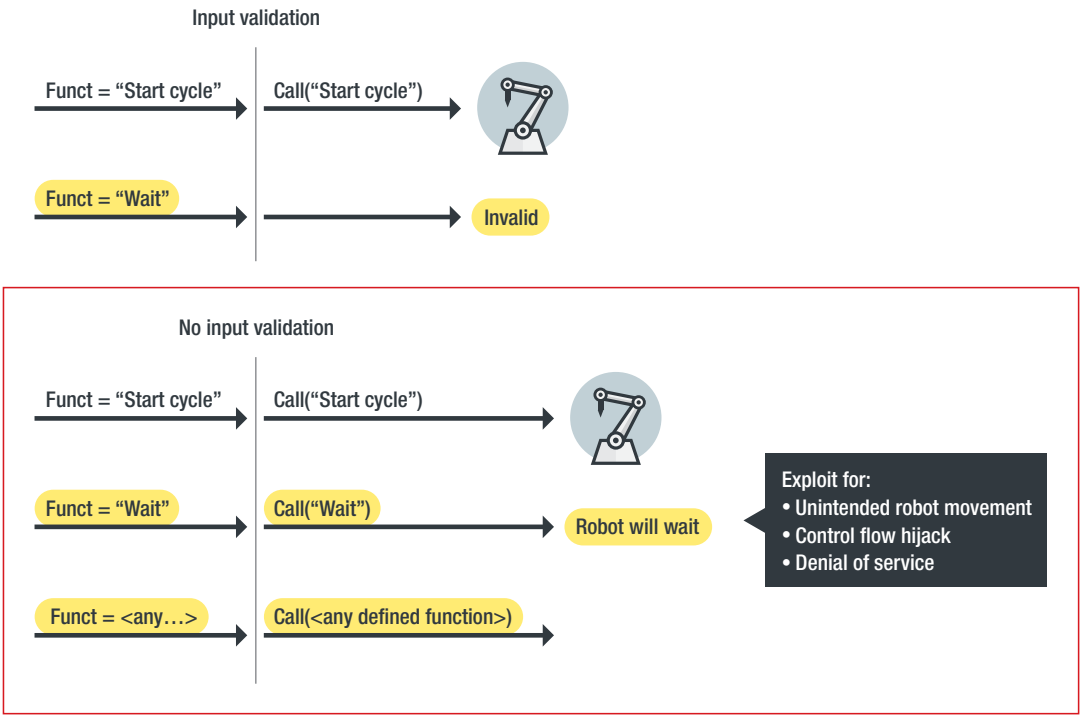


Figure 36. Vulnerable code pattern: From unsanitized (e.g., file, network, serial) data to command invocation

4.5.4 Malicious Code Loading

By combining networking primitives and call by name (i.e., deferred code loading and execution), we verified that it is possible to craft fairly advanced malicious programs. As summarized in Figure 37, the program that we used in our demonstration opens a connection to a remote server, receives the malicious payload, writes it onto a file, executes it (after a delay), and, finally, deletes it once done. An excerpt of an example of this malicious program written in ABB's Rapid is shown in Figure 38. The first three lines of the excerpt regard the robot movement instructions. From the fourth line onward, the malicious part starts receiving data from the network; the data is stored as code, which is dynamically loaded with the load dynamic instruction. The `connect_socket` function shows how the network connectivity can be implemented.

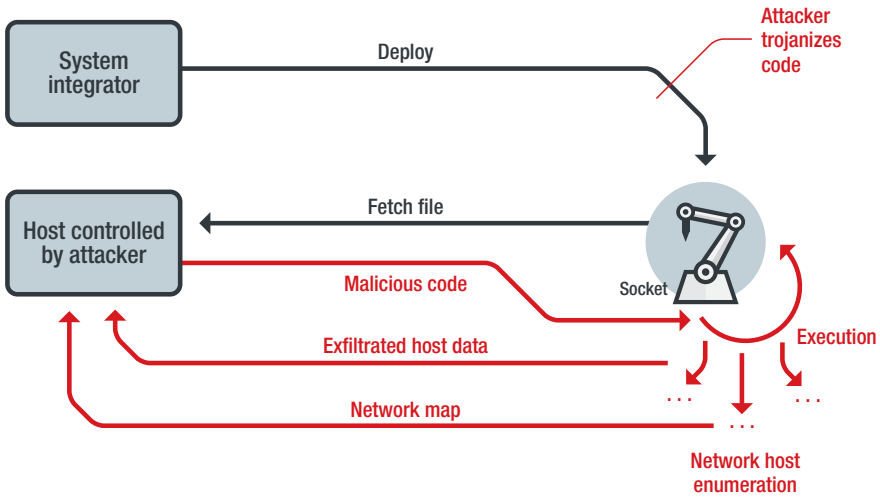


Figure 37. Malicious code pattern: Malicious code loading for creating targeted malware with dropper-like capabilities

```
            ok := StrToVal(tmp, pos{3});
            TPWrite "Moving to: " + ValToStr(pos{1}) + ", " + ValToStr(pos{2}) + ", " +
            move_cmd pos{1}, pos{2}, pos{3};
        ELSEIF StrMatch(data, 1, "load") < length THEN
            ! Remote code loading and execution
            ! message format: "load: name"
            start := StrMatch(data, 1, ":");
            ! remove heading space and trailing newline
            name := StrPart(data, start+2, length - start - 2);
            receivesave name, "!---end---";
            Load \Dynamic, diskhome \File:=name + ".mod";
            %name + ":main"%;
            UnLoad diskhome \File:=name + ".mod";
            RemoveFile diskhome + "/" + name + ".mod";
        ENDIF
    ENDPROC

    PROC connect_socket()
        VAR string clientIP;
        WHILE SocketGetStatus(clientsock) <> SOCKET_CONNECTED DO
            SocketAccept sock, clientsock, \ClientAddress:=clientIP, \Time:=WAIT_MAX;
            IF SocketGetStatus(clientsock) <> SOCKET_CONNECTED THEN
                !//Wait 0.5 seconds for the next reconnection
                WaitTime 0.5;
            ENDIF
        ENDWHILE
    ENDPROC
```

Figure 38. An example of automation logic that embeds a malicious component

From this, the attacker can implement various payloads, depending on the features available on the manufacturing runtime. For example, if the specific programming language has low-level network functionalities available (e.g., open socket), these can be used to enumerate host targets in the same network by trying to connect to well-known ports. If there are low-level file-system access functionalities (e.g., open, read, or write files and directories), these can be used to harvest files on the machine. By combining these two sets of functionalities, an attacker can create over-the-network data exfiltration routines.

Given the interplay between vulnerable and malicious code summarized in Figure 33, instead of the malicious (or compromised) system integrator shown in Figure 37, such a dropper might first infect the target machine via a vulnerability (e.g., the file access case described in Section 4.5.1), and then find further targets by enumerating hosts on the network.

05 Defense and Mitigation

Defending a smart manufacturing system is challenging because the environment itself is complex. Focusing on "keeping attackers out" is clearly important, but this has been the usual advice for decades, regardless of the system. Such an approach is not future-proof because there is a tendency toward increased connectivity and dynamic setups with modular plants that can be reconfigured as needed, as opposed to the classic, static deployments. This has an impact on security policies, which should be moving away from the assumption that every endpoint or machine within a manufacturing plant is trusted, leaving the floor open to a more granular approach. As we have shown, network traffic coming from an industrial robot — to take just one example — may not be coming from trustworthy software because it might be malicious or it could have been exploited. The challenge is that there are currently no simple ways of authenticating and signing the software and data flowing into these complex systems, essentially because not all systems support such security requirements.

5.1 Securing Current Smart Manufacturing Systems

This section complements the attack-specific defense approaches presented in Table 2 and at the beginning of Sections 4.1, 4.2, 4.3, 4.4, and 4.5.

At the network level, deep packet inspection that supports the relevant OT protocols should be implemented to spot anomalous payloads. For endpoints, integrity checks should be run periodically to receive alerts for any altered software component (e.g., an automation script or a file that holds calibration parameters). For IIoT devices, code signing is also required, but it should not be limited to the final firmware; it should also include any other dependencies because we have seen that third-party libraries could also be hiding malicious functionalities.

Risk analysis related to automation software is another important element. Traditional risk analysis in industrial automation settings focuses on safety and sometimes relies on safety mechanisms as a sort of "insurance" or "safety net." However, modern manufacturing machines are evolving in their use of complex safety systems. For example, in systems involving collaborative robots, which work side by side with humans, safety is implemented in the software (at the firmware level), which clearly changes its position in the risk analysis process.

As a general guiding principle, we recommend "thinking outside the box." This paper should serve as a prompt for thinking exercises for both basic and unconventional attack vectors, which are precisely what advanced attackers think about for the attacks that they eventually act on.

5.2 Countermeasures Against Future Threats

What we demonstrate in this paper is what we predict might happen in the future. Our predictions are not based on intuition; all of the technical prerequisites are in place. Before witnessing the first attacks from or involving infected PLCs, only a few people were persuaded that it would be a reality, but now people are dealing with the exploitation of PLCs. Similarly, we hope that this paper can persuade enough key players to believe and invest in the required innovation that is necessary to raise the bar for security.

The first step is to realize that the same level of maturity found in the secure coding practices and defenses of non-OT software (e.g., mobile apps, web apps, cloud environments) must be reached. In the context of web apps as a comparative example, despite the continued existence of input validation vulnerabilities, the overall situation has improved substantially, since it is much harder to find basic input validation bugs today than it was, say, a decade ago. Likewise, despite the continued existence of malware in mobile devices, rudimentary mobile malware has been eliminated. In this paper, we show that we are at a stage where simple input validation vulnerabilities are present in automation code for complex manufacturing machines (e.g., robots) and where malicious or altered extensions (e.g., add-ins, digital twins) can be written and distributed through app stores, much as they were when the very first version of the Google Play Store was initially released. (Malware found in the Google Play Store has become newsworthy on account of its being uncommon, and it takes quite a lot of effort on an attacker's part to evade all of the security checks Google has in place.) Our desire is to see the software delivery mechanisms for OT software being ready for the attacks that we demonstrate in this paper.

The second step is to focus on these three areas to improve current products and embed them with security functionalities:

- A full chain of trust should be properly set up for data and software within a smart manufacturing environment. The code that implements the automation logic, including the firmware running on a custom IIoT device, should not be just "transferred" from the engineering workstation — or worse, the developer's laptop — to the industrial machine. A cryptography-backed code-signing mechanism is required to ensure that all of the libraries incorporated by that code are signed and that the final code is signed as well. This entails support from industrial machine vendors. Similar verification routines should be in place in the development environment: Active software components (e.g., plug-ins, add-ins, extensions) and digital twins must all be signed.

- Detection mechanisms should be in place to recognize vulnerable or malicious logic in complex manufacturing machines (e.g., industrial robots). As we demonstrate in this paper, in addition to the more outright scenario of a malicious developer intentionally creating malicious logic (with time bomb-like behavior), a developer can mistakenly introduce vulnerabilities even in domain-specific languages. Being able to detect these cases will enable short-time prevention because it will alert engineers to the fact that they are about to deploy vulnerable automation logic (e.g., with an "overprivileged" routine or input validation bugs), so that they can take corrective actions. Trend Micro has developed a technology with detection capabilities that can protect its customers from these advanced attacks. Certain portions of this technology are patent pending.

- Sandboxing and privilege separation mechanisms should be implemented for software running on industrial machines and development environments. Mobile apps, for example, must declare and request permission before accessing storage or network resources. So must automation logic and active software components because they are not trustworthy and must never be considered as trusted elements

06 Conclusion

Smart manufacturing systems are designed and deployed under the assumption that they will be isolated from both the outside world and the rest of the corporate network. On one hand, this does not necessarily mean that remote attackers should not be considered: Remote attackers will try alternative, indirect routes (e.g., infected automation logic or software extensions), which we show in this paper to be possible. On the other hand, and perhaps more importantly, the closed-world assumption automatically implies that local attackers have full power: Because of the lack of isolation between the parts of a smart manufacturing system (e.g., all PLCs and machines on the same, flat network), any endpoint will trust any other endpoint and a local attacker will be able to do practically anything they want. We believe this should change.

Attackers are not sitting back and hoping for a high-profile, vulnerable smart manufacturing system to pop up on search engines like Shodan, ready for them to attack. We believe that unconventional attack vectors such as the ones we explore are more likely for an advanced attacker profile. This possibility is increased by the fact that smart manufacturing systems, while made of hardware, live in an ecosystem with an intricate net of interdependencies. Hardware is only one, small part of the equation. There are also other components: software, libraries, developers, business relations, and so on, including software used to develop other software, libraries sold by one company that is used by another company, system integrators who work for several factories. We show how this has repercussions on the types of attacks that are possible in smart manufacturing systems, such as those that involve malicious industrial add-ins and those that trojanize custom IIoT devices.

Once an attacker has landed on a smart manufacturing system, they have unique opportunities for lateral movement, some of which we believe had been unexplored until now. We found security-critical design issues in the automation logic in robots, which not only create ground for vulnerabilities (for which no automated vulnerability scanners exist yet), but also allow the implementation of malicious logic (which will pass undetected, again in the absence of scanners).

We are now ready to answer the research questions stated in Section 1.5. The first, main question was, “Under which threat and attacker models are certain attacks possible, and what are the consequences?” Setting internal attackers aside, we note that external attackers will try to indirectly infect the endpoints through targeted malware. This alone is not surprising; the novel part is that some OT software may offer opportunities for targeting not only one specific person but broader categories of people who all use the same software (e.g., OT developers). This similarly holds true for software libraries used for IIoT development. This answers the second question, “Are there any overlooked vectors that could facilitate an attacker’s getting a foothold in these systems?”

Using such attack vectors, the attacker can gain persistence using, for example, compromised automation logic (e.g., running on industrial robots). The next question follows up on this point: "What is the security impact of modern industrial software development practices, including the use of open libraries, with complex interdependencies?" Programs written in industrial development environments, which do not enforce the use of secure components (e.g., code signing, sandboxing), end up running on a manufacturing machine (e.g., robot). Similarly, IoT firmware that includes complex dependencies and a lot of "unofficial" libraries ends up monitoring or affecting the behavior of the machines. Since all of the components of a smart manufacturing plant are usually connected to the same, flat network, anything can happen. And because an attacker could really do anything to the system, the consequences are difficult to estimate.

The final question was, "What is the cybersecurity awareness level of the technical personnel who engineer, program, and operate in smart manufacturing environments?" Our survey and our analysis of online community discussion groups confirm that people working in OT environments consider security as an "add-on" rather than a process.

A smart manufacturing system does not exist in a vacuum: It is a complex ecosystem of machines, components, and people that can be taken advantage of by threat actors to launch both conventional and unconventional attacks. By shedding light on the different attack vectors that need to be focused on, especially the unconventional ones, we hope that this paper will increase awareness levels, particularly of individuals who are involved in operating smart manufacturing systems.

Appendix

The tactics and techniques used in the attacks discussed in this paper are mapped below using the MITRE ATT&CK® for Industrial Control Systems matrix.[67]

Tactic	Technique	ID	Description
Initial access	Engineering workstation compromise	T818	Used to access the control system applications and equipment.
Persistence	Project file infection	T873	Used to download an infected program to a PLC in the operating environment, enabling further execution and persistence techniques
Discovery	Network service scanning	T1046	Used to get a listing of services running on remote hosts
Impact	Loss of safety	T880	Used to possibly inhibit safety mechanisms that allow the injury and possible loss of life
	Manipulation of Control	T831	Used to communicate with and command physical control processes

Table 3. The tactics and techniques used in the attack involving compromise through a malicious industrial add-in

Tactic	Technique	ID	Description
Initial access	Supply chain compromise	T862	Used to gain access to the control system environment
Persistence	Module firmware	T839	Used to install malicious or vulnerable firmware onto modular hardware devices
Inhibit response function	Denial of service	T814	Used to disrupt expected device functionality
Impact	Denial of control	T813	Used to deny process control access to cause temporary loss of communication with the control device
	Loss of control	T827	Used to achieve sustained loss of control
	Loss of productivity and revenue	T828	Used to cause loss of productivity and revenue
	Loss of safety	T880	Used to possibly inhibit safety mechanisms that allow the injury and possible loss of life

Table 4. The tactics and techniques used in the attack involving trojanization of a custom IIoT device

Tactic	Technique	ID	Description
Execution	Graphical user interface	T1061	Used to search for information and execute files
Lateral movement	Default credentials	T812	Used to abuse default credentials that have not been properly modified or disabled
Impact	Loss of safety	T880	Used to possibly inhibit safety mechanisms that allow the injury and possible loss of life
	Manipulation of control	T831	Used to communicate with and command physical control processes
	Theft of operational information	T882	Used to steal operational information on a production environment

Table 5. The tactics and techniques used in the attack involving the exploitation of a vulnerable mobile HMI

Tactic	Technique	ID	Description
Initial access	Data historian compromise	T810	Used to gain a foothold in the control system environment
Impair process control	Modify parameter	T836	Used to produce an outcome outside of what was intended by the operators
Impact	Denial of control	T813	Used to deny process control access to cause temporary loss of communication with the control device
	Loss of safety	T880	Used to possibly inhibit safety mechanisms that allow the injury and possible loss of life
	Manipulation of control	T831	Used to communicate with and command physical control processes

Table 6. The tactics and techniques used in the attack involving data mangling on the MES

Tactic	Technique	ID	Description
Initial access	Engineering workstation compromise	T818	Used to access the control system applications and equipment
	Supply chain compromise	T862	Used to gain access to the control systems environment
Persistence	Project file infection	T873	Used to download an infected program to a PLC in the operating environment, enabling further execution and persistence techniques
Collection	Program upload	T845	Used to gather information about an industrial process
Impact	Loss of productivity and revenue	T828	Used to cause loss of productivity and revenue
	Manipulation of view	T833	Used to manipulate the information reported back to the operators or controllers

Table 7. The tactics and techniques used in the attack involving the use of the vulnerable or malicious automation logic in a complex manufacturing machine

References

1 Mohit Kumar. (7 August 2018). *The Hacker News*. "TSMC Chip Maker Blames WannaCry Malware for Production Halt." Last accessed on 25 February 2020 at https://thehackernews.com/2018/08/tsmc-wannacry-ransomware-attack.html.

2 Sophos. (19 September 2017). *Naked Security by Sophos*. "PyPI Python repository hit by typosquatting sneak attack." Last accessed on 24 February 2020 at https://nakedsecurity.sophos.com/2017/09/19/pypi-python-repository-hit-by-typosquatting-sneak-attack/.

3 Brian Gorenc and Fritz Sands. (23 May 2017). *Trend Micro*. "Hacker Machine Interface." Last accessed on 25 February 2020 at https://documents.trendmicro.com/assets/wp/wp-hacker-machine-interface.pdf.

4 Nilufer Tuptuk and Stephen Hailes. (April 2018). *UCL Discovery*. "Security of smart manufacturing systems." Last accessed on 17 March 2020 at https://discovery.ucl.ac.uk/id/eprint/10051762/1/1-s2.0-S0278612518300463-main.pdf.

5 Siemens. (n.d.). *Siemens*. "Totally Integrated Automation – Future inside." Last accessed on 24 February 2020 at https://new.siemens.com/global/en/products/automation/topic-areas/tia.html.

6 Politecnico di Milano. (n.d.). *Industry 4.0 Lab*. "A Teaching and Research Lab for Industry 4.0." Last accessed on 24 February 2020 at https://www.industry40lab.org/.

7 Politecnico di Milano. (n.d.). *Politecnico Milano School of Management*. "Politecnico Milano School of Management." Last accessed on 27 February 2020 at https://www.som.polimi.it/.

8 Magnus Åkerman. (June 2018). *ResearchGate*. "Implementing Shop Floor IT for Industry 4.0." Last accessed on 24 February 2020 at https://www.researchgate.net/publication/326224890_Implementing_Shop_Floor_IT_for_Industry_40.

9 Trend Micro Research. (3 April 2019). *Trend Micro*. "Security in the Era of Industry 4.0: Dealing With Threats to Smart Manufacturing Environments." Last accessed on 6 March 2020 at https://www.trendmicro.com/vinfo/us/security/news/internet-of-things/security-in-the-era-of-industry-4-dealing-with-threats-to-smart-manufacturing-environments.

10 Hannover Messe. (n.d.). *Hannover Messe*. "Hannover Messe." Last accessed on 27 February 2020 at https://www.hannovermesse.de/en/expo/exhibitor-short-index/index-2.

11 Politecnico di Milano. (n.d.). *The Fenix Project*. "The Fenix Project." Last accessed on 27 February 2020 at http://www.fenix-project.eu/.

12 European Commission. (n.d.). *European Commission*. "What is Horizon 2020?" Last accessed on 18 March 2020 at https://ec.europa.eu/programmes/horizon2020/en/what-horizon-2020.

13 Siemens. (n.d.). *Siemens*. "Simatic DP CPU 1510SP-1 PN." Last accessed on 27 February 2020 at https://mall.industry.siemens.com/mall/en/us/Catalog/Product/6ES75101DJ000AB0.

14 Siemens. (n.d.). *Siemens*. "Simatic HMI TP700 Comfort." Last accessed on 27 February 2020 at https://support.industry.siemens.com/cs/pd/127118.

15 Mitsubishi Electric. (n.d.). *Mitsubishi Electric*. "Melfa V-2AJ." Last accessed on 27 February 2020 at https://www.mitsubishielectric.com/fa/products/rbt/robot/pmerit/vertical/fseries/index.html.

16 Siemens. (n.d.). *Siemens*. "Scalance X208." Last accessed on 27 February 2020 at https://support.industry.siemens.com/cs/pd/501190.

17 Ian Verhappen. (17 April 2019). *Control Global*. "What's holding back mobile HMI?" Last accessed on 24 February 2020 at https://www.controlglobal.com/articles/2019/whats-holding-back-mobile-hmi/.

18 Control.com. (n.d.). *Control.com*. "Forum." Last accessed on 31 August 2019 at https://control.com/forums/.

19 PLC.MyForum.ro. (n.d.). *PLC.MyForum.ro*. "PLC.MyForum.ro." Last accessed on 31 August 2019 at http://plc.myforum.ro/.

20 Mr.PLC. (n.d.). *Mr.PLC*. "Forums." Last accessed on 31 August 2019 at http://forums.mrplc.com/.

21 Robotforum. (n.d.). *Robotforum*. "Forum." Last accessed on 31 August 2019 at https://www.robot-forum.com/robotforum/.

22 Reddit. (n.d.). *Reddit*. "Robotics." Last accessed on 31 August 2019 at https://www.reddit.com/r/robotics/.

23 Adam Forum. (n.d.). *Adam Forum*. “Industrial Automation Discussion.” Last accessed on 31 August 2019 at https://forum.adamcommunity.com/index.php.

24 Automation Forum. (n.d.). *Automation Forum*. “Automation Forum.” Last accessed on 31 August 2019 at https://automationforum.in/.

25 DoF. (n.d.). *DoF*. “DoF.” Last accessed on 31 August 2019 at https://dof.robotiq.com/.

26 ABB Robotics. (n.d.). *ABB Robotics*. “User Forum.” Last accessed on 31 August 2019 at https://forums.robotstudio.com/.

27 Universal Robots. (n.d.). *Universal Robots*. “Forum.” Last accessed on 31 August 2019 at https://forum.universal-robots.com/.

28 SolicPLC. (n.d.). *SolisPLC*. “Forum.” Last accessed on 31 August 2019 at https://solisplc.com/forum/.

29 Harry Garrood. (12 July 2019). *Harry Garrood*. “Malicious code in the purescript npm installer.” Last accessed on 24 February 2020 at https://harry.garrood.me/blog/malicious-code-in-purescript-npm-installer/.

30 Federico Maggi, Davide Quarta, Marcello Pogliani, Mario Polino, Andrea M. Zanchettin, and Stefano Zanero. (3 May 2017). *Trend Micro*. “Rogue Robots: Testing the Limits of an Industrial Robot’s Security.” Last accessed on 24 February 2020 at https://documents.trendmicro.com/assets/wp/wp-industrial-robot-security.pdf.

31 Catalin Cimpanu. (7 July 2019). *ZDNet*. “Canonical GitHub account hacked, Ubuntu source code safe.” Last accessed on 24 February 2020 at https://www.zdnet.com/article/canonical-github-account-hacked-ubuntu-source-code-safe/.

32 Bertus. (21 October 2018). *Bertus*. “Cryptocurrency Clipboard Hijacker Discovered in PyPI Repository.” Last accessed on 24 February 2020 at https://medium.com/@bertusk/cryptocurrency-clipboard-hijacker-discovered-in-pypi-repository-b66b8a534a8.

33 Sophos. (19 September 2017). *Naked Security by Sophos*. “PyPI Python repository hit by typosquatting sneak attack.” Last accessed on 24 February 2020 at https://nakedsecurity.sophos.com/2017/09/19/pypi-python-repository-hit-by-typosquatting-sneak-attack/.

34 Curtis Franklin Jr. (27 June 2019). *Dark Reading*. “How Hackers Infiltrate Open Source Projects.” Last accessed on 24 February 2020 at https://www.darkreading.com/application-security/how-hackers-infiltrate-open-source-projects-/d/d-id/1335072.

35 Tara Seals. (2 April 2019). *Threatpost*. “ThreatList: Half of All Attacks Aim at Supply Chain.” Last accessed on 24 February 2020 at https://threatpost.com/half-all-attacks-supply-chain/143391/.

36 Carbon Black. (April 2019). *Carbon Black*. “Carbon Black’s Global Incident Response Threat Report: The Ominous Rise of “Island Hopping” & Counter Incident Response Continues.” Last accessed on 24 February 2020 at https://www.carbonblack.com/global-incident-response-threat-report/april-2019/#form.

37 Claud Xiao. (21 September 2015). *Unit 42*. “More Details on the XcodeGhost Malware and Affected iOS Apps.” Last accessed on 25 February 2020 at https://unit42.paloaltonetworks.com/more-details-on-the-xcodeghost-malware-and-affected-ios-apps/.

38 ABB Robotics. (n.d.). *RobotStudio*. “RobotStudio.” Last accessed on 27 February 2020 at https://robotapps.robotstudio.com/.

39 Intelligent Plant. (n.d.). *Industrial App Store*. “Industrial App Store.” Last accessed on 27 February 2020 at https://appstore.intelligentplant.com/.

40 OrangeApps. (n.d.). *OrangeApps*. “OrangeApps.” Last accessed on 11 March 2020 at https://www.orangeapps.de/?lng=en#appstore.

41 Siwiat. (n.d.). *Siwiat App-Store*. “Siwiat App-Store.” Last accessed on 27 February 2020 at https://siwiat.com/en/app/app-store/.

42 Arduino. (n.d.). *Arduino*. “Arduino Industrial 101.” Last accessed on 27 February 2020 at https://store.arduino.cc/usa/arduino-industrial-101.

43 Industrial Shields. (n.d.). *Industrial Shields*. “Industrial Shields.” Last accessed on 27 February 2020 at https://www.industrialshields.com/.

44 Industruino. (n.d.). *Industruino*. “Industruino.” Last accessed on 27 February 2020 at https://industruino.com/.

45 Sfera Labs. (n.d.). *Iono Arduino*. “Iono Arduino.” Last accessed on 27 February 2020 at https://www.sferalabs.cc/iono-arduino/.

46 Siemens. (n.d.). *Siemens*. “Simatic IoT2000.” Last accessed on 27 February 2020 at https://new.siemens.com/global/en/products/automation/pc-based/iot-gateways/iot2000.html.

47 Sergio Pastrana, Jorge, Rodriguez-Canseco, and Alejandro Calleja. (2016). *Jornadas Nacionales de Investigación en Ciberseguridad (JNIC)*. “ArduWorm: A Functional Malware Targeting Arduino Devices.” Last accessed on 25 February 2020 at https://evosec.eu/wp-content/uploads/2016/11/2016jnic1.pdf.

48 Aurélien Francillon and Claude Castellucciaa. (22 January 2009). *Arxiv.org*. “Code injection attacks on harvard-architecture devices.” Last accessed on 25 February 2020 at https://arxiv.org/abs/0901.3482.

49 NASA Office of Inspector General. (18 June 2019). *National Aeronautics and Space Administration*. “Cybersecurity Management and Oversight at the Jet Propulsion Laboratory.” Last accessed on 25 February 2020 at https://oig.nasa.gov/docs/IG-19-022.pdf.

50 Luca Bongiorni. (16 August 2019). *Luca Bongiorni*. “USBSamurai — A Remotely Controlled Malicious USB HID Injecting Cable for less than 10$.” Last accessed on 25 February 2020 at https://medium.com/@LucaBongiorni/usbsamurai-a-remotely-controlled-malicious-usb-hid-injecting-cable-for-less-than-10-ebf4b81e1d0b.

51 Brian Gorenc and Fritz Sands. (23 May 2017). *Trend Micro*. “Hacker Machine Interface.” Last accessed on 25 February 2020 at https://documents.trendmicro.com/assets/wp/wp-hacker-machine-interface.pdf.

52 Qcadoo. (n.d.). *Qcadoo*. “Qcadoo.” Last accessed on 27 February 2020 at https://www.qcadoo.com/.

53 Factority. (n.d.). *Factority*. “Factority.” Last accessed on 27 February 2020 at https://www.openhub.net/p/factority.

54 Gartner. (n.d.). *Gartner*. “Manufacturing Execution Systems Market.” Last accessed on 27 February 2020 at https://www.gartner.com/reviews/market/manufacturing-execution-systems.

55 ROS-Industrial. (n.d.). *ROS-Industrial*. “ROS-Industrial Consortium.” Last accessed on 27 February 2020 at https://rosindustrial.org/ric/current-members/.

56 Federico Maggi, Davide Quarta, Marcello Pogliani, Mario Polino, Andrea M. Zanchettin, and Stefano Zanero. (3 May 2017). *Trend Micro*. “Rogue Robots: Testing the Limits of an Industrial Robot’s Security.” Last accessed on 24 February 2020 at https://documents.trendmicro.com/assets/wp/wp-industrial-robot-security.pdf.

57 Trend Micro. (13 May 2014). *Trend Micro*. “Stuxnet.” Last accessed on 11 March 2020 at https://www.trendmicro.com/vinfo/us/threat-encyclopedia/malware/stuxnet.

58 ICS-CERT. (7 April 2020). *Cybersecurity and Infrastructure Security Agency (CISA)*. “ICS Advisory (ICSA-20-098-05).” Last accessed on 14 April 2020 at https://www.us-cert.gov/ics/advisories/icsa-20-098-05.

59 Let’s Encrypt. (n.d.). *Let’s Encrypt*. “Let’s Encrypt.” Last accessed on 12 March 2020 at https://letsencrypt.org/.

60 Arduino. (n.d.). *Arduino*. “Libraries.” Last accessed on 26 February 2020 at https://www.arduino.cc/en/reference/libraries.

61 Federico Maggi, Davide Quarta, Marcello Pogliani, Mario Polino, Andrea M. Zanchettin, and Stefano Zanero. (3 May 2017). *Trend Micro*. “Rogue Robots: Testing the Limits of an Industrial Robot’s Security.” Last accessed on 24 February 2020 at https://documents.trendmicro.com/assets/wp/wp-industrial-robot-security.pdf.

62 Harry Garrood. (12 July 2019). *Harry Garrood*. “Malicious code in the purescript npm installer.” Last accessed on 24 February 2020 at https://harry.garrood.me/blog/malicious-code-in-purescript-npm-installer/.

63 Catalin Cimpanu. (7 July 2019). *ZDNet*. “Canonical GitHub account hacked, Ubuntu source code safe.” Last accessed on 24 February 2020 at https://www.zdnet.com/article/canonical-github-account-hacked-ubuntu-source-code-safe/.

64 Sophos. (19 September 2017). *Naked Security by Sophos*. “PyPI Python repository hit by typosquatting sneak attack.” Last accessed on 24 February 2020 at https://nakedsecurity.sophos.com/2017/09/19/pypi-python-repository-hit-by-typosquatting-sneak-attack/.

65 PlatformIO. (n.d.). *PlatformIO*. “Libraries.” Last accessed on 27 February 2020 at https://platformio.org/lib/search.

66 Marcello Pogliani, Davide Quarta, Mario Polino, Martino Vittone, Federico Maggi, and Stefano Zanero. (13 February 2019). *Journal of Computer Virology and Hacking Techniques*. “Security of controlled manufacturing systems in the connected factory: the case of industrial robots.” Last accessed on 25 March 2020 at https://link.springer.com/article/10.1007/s11416-019-00329-8.

67 MITRE. (4 March 2020). *MITRE*. “ATT&CK® for Industrial Control Systems.” Last accessed on 25 March 2020 at https://collaborate.mitre.org/attackics/index.php/Main_Page.

TREND MICRO™ RESEARCH

Trend Micro, a global leader in cybersecurity, helps to make the world safe for exchanging digital information.

Trend Micro Research is powered by experts who are passionate about discovering new threats, sharing key insights, and supporting efforts to stop cybercriminals. Our global team helps identify millions of threats daily, leads the industry in vulnerability disclosures, and publishes innovative research on new threat techniques. We continually work to anticipate new threats and deliver thought-provoking research.

www.trendmicro.com

TREND MICRO™ | research