A Security Evaluation of Industrial Radio Remote Controllers

Federico Maggi, Marco Balduzzi, Jonathan Andersson, Philippe Lin
Stephen Hilt, Akira Urano, and Rainer Vosseler

Trend Micro Research

Abstract Heavy industrial machinery is a primary asset for the operation of key sectors such as construction, manufacturing, and logistics. Targeted attacks against these assets could result in incidents, fatal injuries, and substantial financial loss. Given the importance of such scenarios, we analyzed and evaluated the security implications of the technology used to operate and control this machinery, namely industrial radio remote controllers. We conducted the first-ever security analysis of this technology, which relies on proprietary radio-frequency protocols to implement remote-control functionalities. Through a two-phase evaluation approach we discovered important flaws in the design and implementation of industrial remote controllers. In this paper we introduce and describe 5 practical attacks affecting major vendors and multiple real-world installations. We conclude by discussing how a challenging responsible disclosure process resulted in first-ever security patches and improved security awareness.

1 Introduction

Industrial applications such as mining, construction, manufacturing, and logistics are fundamental for any society. These sectors offer very attractive targets for criminals, and the consequences of attacks can be severe and easily produce substantial financial loss, injuries, if not fatalities. In 2013, for example, a *failure* (not an attack) caused a heavy load to fall during a refueling outage and caused the automatic shutdown of a power plant reactor in Arkansas [2]. In 2017 [7] about 80 among ports and terminals worldwide experienced significant delays and downtime due to the first publicly-reported attack against maritime facilities, which resulted in about $300 million loss. Cases like Havex [24], Black Energy [22], Industroyer [12], and Triton [20], show that attackers are very interested in critical industry sectors in general.

Although IT security is a well-studied research field, industrial IoT and operational technology (OT) security research have received less attention from the academic community. In this research, we analyzed how industrial machinery such as overhead cranes (used to move heavy loads in factories), drillers (employed to dig tunnels), and similar, are controlled and operated. Notably, these machines are often maneuvered remotely via radio frequency, by means of so-called industrial remote controllers. These are are hand-held devices, with keypads or joysticks, which the operator uses to issue commands to the controlled machine.

These controllers are thus critical for those factories and industries that employ them to fulfill their business needs. Given their importance, we analyzed how these systems are designed, implemented, and deployed, assuming the presence of active attackers.

Through a market survey to identify vendors, technologies, and standards at play, we discovered 17 vendors that globally distribute millions of industrial remote controllers since more than 30 years (Section 2). Starting from the technical documentation obtained from the vendors' websites and other sources, we analyzed the attack surface and defined the attacker model (Section 3.1).

We then conducted a two-phase security evaluation. First, we analyzed in-depth the remote controllers from two independent vendors (Section 4.1 and 4.2). We reverse engineered their internals from a hardware and software perspective. Based on this analysis, we designed and implemented five attacks (described in Section 3.2), and tested how a miscreant could exploit them to subvert the controlled machine or create unsafe conditions. Secondly, we extended our analysis to on-site deployments to verify that our attacks apply to real-world installations (Section 4.3). Indeed, the deployments that we tested did not include any further countermeasures against our attacks. Overall, we visited 24 facilities including manufacturing plants, transportation hubs, and construction sites, totaling 7 vendors and 14 remote controllers tested.

We conclude our work by designing and building a coin-sized, portable device that implements our attacks (Section 5). This shows that an attacker with just temporary access to a facility could permanently control the target industrial machines from any location, beyond the radio-frequency range.

Overall, the contributions of our work are as follows:

- We investigated the security of radio remote controllers, which represent a core technology for controlling and automating heavy industrial machinery like cranes, hoists, and drilling rigs. To the best of our knowledge, we are the first to look into this direction.
- We conducted a two-phase security evaluation that highlights important security flaws in the design, implementation, and deployment of these devices. We describe 5 attacks and verify their practical feasibility on real-world installations.
- We document the lesson learned from the rather long responsible-disclosure process, which concluded with first-ever security patches and improved security awareness (Section 6).
- We release the tools developed within this project, in particular RFQuack[1], with the hope of providing additional benefit to the research community.

2 Background on Industrial Radio Remote Controllers

Industrial radio remote controllers are a popular and well-established technology used to operate and automate heavy machinery in the mining, material-handling,

[1] An Arduino-based open-hardware/software research framework to analyze sub-GHz radio protocols: `https://github.com/trendmicro/rfquack`

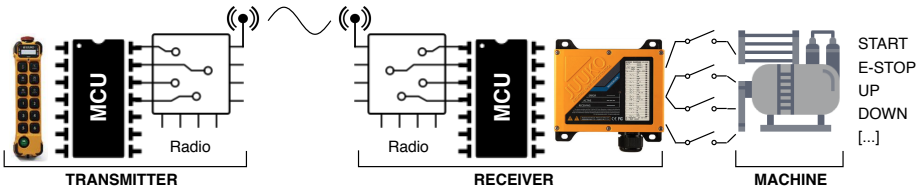

Figure 1. High-level architecture of an industrial RF controller.

and construction sectors, among others. An industrial remote controller system is comprised of one transmitter (TX) and one receiver (RX)[2]. An operator uses the TX to remotely send commands to the RX installed on the controlled machine, as depicted in Figure 1. The RX communicates with the machine via relay switches or serial (industrial) protocols (e.g., RS485, ModBus, Profibus, DeviceNet, CAN[3]). In principle, any electrical load could be controlled. In the typical scenario, a worker in a construction site operates a hand-held keypad to maneuver a crane tower and lift heavy items. For each of these operations, the TX encodes a command and transmits it over the air using radio-frequency (RF) protocols. Each command is then translated by the RX into a physical actuation (e.g., open-close a switch).

Industrial radio remote controllers typically operate in the sub-GHz bands that are reserved for industrial, scientific, and medical applications (i.e., 315, 433, 868, and 915 MHz). Most importantly, these remote controllers do not appear to adhear to any standard communication protocols. A few exceptions apart[4], they all rely on proprietary protocols with little or no security features. We draw this conclusion from our experience on real systems, and after having analyzed the public documentation obtained from the vendors' websites or provided by the Federal Communications Commission (FCC), when available[5]. Although few controllers operate in the 2.4 GHz band, we focused our research on those working in the sub-GHz bands.

Through a market survey, we found 17 vendors[6] that distribute worldwide, some of which on the market for more than 30 years. The price tag for these systems is around $300–1,500. Considering these figures, their annual revenue and sales statistics, we could roughly estimate that millions of units are installed worldwide.

[2] Multi transmitter and multi receiver scenarios are possible.

[3] For example: `http://www.hetronic.com/Control-Solutions/Receivers/Serial-Communication`

[4] Liebherr and Schneider Electric use Bluetooth Low Energy (BLE).

[5] Searchable FCC ID database at `https://fccid.io`

[6] Autec (established in 1986), Hetronic (1982), Saga (1997), Circuit Design (1974), Elca (1991), Telecrane (1985), Juuko (1994), HBC-radiomatic (1947), Cattron (1946), Tele Radio (1955), Scanreco (1980), Shanghai Techwell Autocontrol Technology (2005), Remote Control Technology (1982), Akerstroms (1918), Jay Electronique (1962), Itowa (1986), 3-Elite (1995)

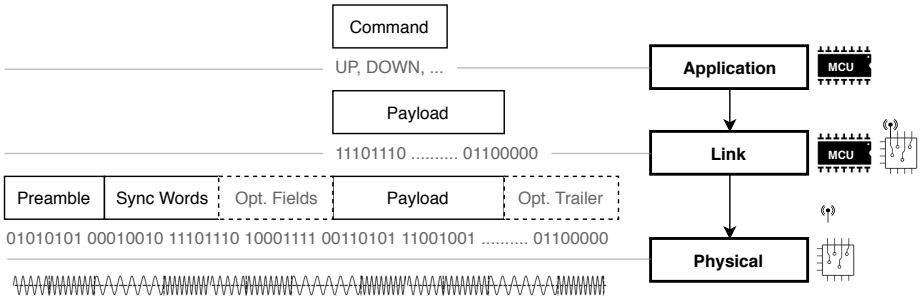

Figure 2. Each command is encoded in a custom way into the payload, which is encapsulated in a radio packet. The preamble "wakes up" the RX to consider the following data, and allows the calculation of the symbol length; the receiver compares the sync words to a reference value to confirm the symbol length and ensure that the upcoming data is a valid packet. The position and presence of other fields such as address and length vary across vendors.

2.1 Controllers Overview

Internally, RX and TX are very similar, as depicted in Figure 1. They both have a main printed circuit board (PCB) equipped with one or more micro controller units (MCUs), and a set of inputs (e.g., switches and buttons on the TXs) and outputs (e.g., relay switches or other buses on the RXs).

In modern controllers the radio is soldered on a detachable daughter board, which can be easily removed and replaced for easy maintenance. Older generation systems (yet still supported and deployed) are composed of a single PCB that includes the radio.

These systems are full-fledged embedded systems with programming interfaces and fairly powerful MCUs, OLED displays, and so on. Their code (firmware) typically takes a few kilobytes of memory and is responsible of handling interrupts from the inputs (including those received from the radio), deciding on the action, and prepare the message to be sent via radio.

2.2 Communication Protocol

In simplified terms, TX and RX communicate by exchanging *commands* (see Figure 2). Typical commands are 'up', 'down', 'left', 'right', or 1, 2, 3, and so on. These commands are mapped onto inputs (e.g., buttons on the TX side) and outputs (e.g., relays on the RX side).

As for other RF protocols, these devices implement basic communication features such as addressing, encoding, and encapsulation. When a button is pressed on the TX, the *application* layer encodes the command in the *payload*, a sequence of bytes. Additional control data like pairing code, packet length and checksum are normally added to the resulting packet, which is passed to the *link* layer. This layer adds RF-specific fields (e.g., preamble, trailer, radio checksums)

and encodes the resulting data (e.g., with Manchester) for better signal-to-noise ratio.

At this point, the packet is transferred from the MCU to the radio subsystem. Although modern radio chips expose quite advanced and high-level packet-handling primitives, the essential part of the radio subsystem is a digital RF modem, which implements the *physical* layer. For transmission, the logical symbols (e.g., the zeros and ones of the payload) are translated into one or more properties of the carrier signal (e.g., frequency offset, amplitude level). For instance, frequency shift keeying (FSK) represents each of the n symbols (`0` and `1` if $n = 2$) by shifting the frequency by 2 deviations around the carrier, whereas amplitude shift keeying (ASK) applies the same concept by varying the signal amplitude. This process is called *modulation*. In reception, the same operations are executed in reverse order (i.e., the signal is de-modulated).

From the experience we gathered in analyzing these radio remote controllers, and the code review we did of a popular radio hardware abstraction library (RadioHead[7]), we can reasonably assume the generic packet structure depicted in Figure 2.

2.3 Safety Features and Security Model

Safety is of paramount importance and main focus of these systems' design. For example, the RX has so-called *safety relays* that disable the main load of the controlled machine. Operators are trained to trigger these safety relays whenever needed, by pushing an emergency button (e-stop) on the TX. This transmits an emergency-stop command. When the RX receives an e-stop command it disengages the safety relays. During our on-site evaluation (Section 4.3), we learned that the e-stop button is also used to simply turn off the machine.

According to our analysis, *security* is not taken into account in the design of these devices. In fact, security comes as a positive "side effect" of the following *safety* features, which are often required by country regulations such as the ISO 13849 or IEC 61508.

Pairing Mechanism TX and RX are paired with a *fixed pairing code*, which is used as an authentication factor (long-term secret). The goal is not to hinder an active attacker, but to allow the concurrent use of multiple radio remote controllers without interferences. Clearly, knowledge of the pairing code allows complete impersonation of a legitimate TX or RX.

Passcode Protection The operator needs to enter a passcode through buttons on the TX, which in turn sends a command that enables the safety relays on the RX. Assuming the secrecy of the passcode, this measure prevents unauthorized personnel from operating the controlled machine *via that transmitter* (not via other means).

[7] `https://www.airspayce.com/mikem/arduino/RadioHead/`

Authorization and Access Control The TX can implement an access-control system that selectively enables or disables features per operator (e.g., by using RFIDs). This prevents inexperienced operators from issuing other commands that could cause injuries under certain conditions.

Virtual Fencing TX and RX communicate via an out-of-band channel (usually infrared) in addition to RF. When the TX is out of sight, the RX does not accept any commands sent via RF. The assumption is that the adversary is unable to control both the channels simultaneously.

3 Security Analysis

Although we followed an empirical approach, driven by access to devices and technical material, we hereby provide a high-level overview of our analysis. We present 3 attacker models and 5 attacks. We defer details and findings to Section 4, which describes how we tested—in lab and on site—remote controllers from 7 vendors for the vulnerabilities required to implement these attacks.

3.1 Attacker Models

Our analysis considered three types of attackers.

Attacker within RF Range This attacker operates within the RF range of the target device (about 300 meters according to our on-site tests), and wants to tamper with the RF protocol of the industrial controllers. For instance, from outside a factory, the attacker could target and reach the machinery located inside.

Attacker with Local Network Access This attacker has access to the computer used to re-configure or re-program the controllers. Most vendors sell software and programming dongles—that can be installed on any computer—to effectively enable system integrators, distributors, resellers, and even end users, customize their controllers. One vendor even offers pre-configured tablets to let its clients customize their devices[8]. This indicates user readiness and demand, and an additional attack surface. Alarmingly, the programming software that we analyzed run on unsupported operating systems (e.g., Windows XP); thus is not unlikely to assume that an attacker may already have a foothold in these systems.

Sophisticated, Stealthy Attacker This attacker is able to walk, drive or fly within the RF range for a few seconds only—long enough to drop or hide a coin-sized, battery-operated (malicious) RF transceiver featuring a cellular modem for remote access, such as the one described in Section 5. Ten years ago, this scenario would be unrealistic and expensive, whereas nowadays the convergence and ubiquity of cheap hardware, embedded software, and radio technology has lowered the barrier significantly.

[8] `https://www.telecrane.it/en/tablet-with-original-software/`

The Role of Software-defined Radios Regardless of the attacker type, the entry barrier to RF threats has lowered over the past ten years. With little investment (e.g., $100–500), anyone can leverage software-defined radios (SDRs) to implement a wide variety of (benign and malicious) RF tools. SDRs give access to the physical layer: They take samples of the signals received by the antenna at very high resolution (e.g., millions of samples per second), and let the user program the rest of the (software-defined) stack, aided by frameworks like GNU Radio [5] or Universal Radio Hacker (URH) [17], just to name two popular ones.

3.2 Attacks Against the Communication Protocol

We distinguish between *attack primitives* (i.e., the smallest self-contained "units" of an attack) and *second-level attacks*, obtained by composition of primitives.

Replay Attack (Primitive) The simplest primitive consists in using a SDR to capture the RF signals during the transmission of legitimate commands (e.g., 'right'), and re-play the captures to impersonate the original TX. With weak or no message-integrity and authentication functionality, every command would result in the very same signal. Thus, a RX could not tell whether a transmission is replayed or original.

Preparing replay attacks is not trivial. The attacker must stay within the RF range of the *target* RX and TX long enough to have the chance to capture all the transmissions needed. Since each paired TX and RX have a unique paring code, the replayed captures are accepted by the RX only if recorded from a paired TX. Most importantly, the attacker must be able to tell which command corresponds to which transmission. In summary, the degree of control depends on the availability of usable transmissions.

Unlike standard protocols like BLE, which are immune against replay attacks, our analysis reveals the protocols used by industrial remote controllers rely on "security through obscurity."

Arbitrary Packet Forgery (Primitive) An attacker who wants to *forge* arbitrary packets—or *manipulate* existing ones to the desired level of control—must have reverse engineered the communication protocol, and found weaknesses in it.

As we describe in Section 4, protocol reverse engineering is needed to understand the RF parameters (e.g., carrier, modulation), as well as the packet structure, data encoding, and so on. This requires the attacker to obtain a device of the same brand and model of the target one.

For example, during our evaluation we discovered that *partial* reverse engineering is sufficient to achieve *full* control over one of the devices (Section 4.2). We just need to assume that the attacker could intercept *one* transmission (any packet would do) from the target system, without even knowing which command it encodes. All the attacker needs to do is manipulate the packet such that to encode *another arbitrary* command, and transmit the packet so obtained.

If the attacker gains *complete* knowledge of the protocol, full packet forgery is possible without further assumptions, as we show in Section 4.1.

E-stop Abuse (Second Level) The emergency stop functionality, which is mandatory in most countries, can be abused to force the controlled machine in a persistent denial-of-service state, opening the floor to extortion threats.

Using any of the two aforementioned primitives, this attack consists in repeatedly transmitting an e-stop command, so that the safety relays would cut the power from the controlled load. Consider that these attacks could be carried out from outside the victim's premises or with a small transceiver hidden very close to the RX, and thus very hard to trace.

Pairing Hijacking (Second Level) While replay attacks are based on repeating *individual* commands, the ability of *fully* impersonating a TX gives the attacker further advantages.

Many industrial controllers come with a "pairing feature," which allows multiple independent TXs to control one RX. Normally, a special sequence of buttons is used to initiate the pairing of a new device. From a RF perspective, during pairing one or more packets are exchanged between TX and RX. If weak or no integrity and authentication are implemented, an attacker could use one of the aforementioned primitives to forge pairing packets, thus mimicking a fictitious TX. As a result, the miscreant would obtain full control.

3.3 Local Network Attack

Depending on the protection level, we distinguish between the ability to reconfigure and re-program the target device.

Malicious Re-configuration and Re-programming Together with the controller, vendors usually provide a programming software to customize button mappings or pairing IDs, or update the firmware of the device. The device needs to be connected to a computer via serial interface (e.g., USB dongle).

If designed and implemented securely, the software is expected to require user confirmation, on both sides (device and computer), before trusting the serial connection. Failing to do so would allow malicious re-configuration or, worse, re-flashing the MCU memory with malicious firmware, under a network-attacker assumption. It is not unrealistic to assume that such attacker could compromise the computer connected to the controller, as the programming software that we analyzed appeared very "legacy," and in some cases required unsupported—and thus vulnerable—systems like Windows XP.

4 Experimental Evaluation

We followed two strategies to verify the feasibility of the aforementioned attacks under different attacker-model assumptions. We wanted to test for both the attack primitives, and for the second-level attacks.

We first emulated an attacker with zero knowledge of the system and verified the vulnerability to the replay-attack primitive, and to e-stop abuse and pairing hijacking—first in lab and then on site. On-site tests aimed at verifying that the

Table 1. Summary of the in-lab evaluation.

Vendor	SAGA (Taiwan, Worldwide)	Juuko (Taiwan, Worldwide)
Tested devices	2	1
RF Attack Primitives		
Replay Attack	✓	✓
Arbitrary Packet Manipulation	✓	✓
Second-level RF Attacks		
E-stop Abuse	✓	✓
Pairing Hijacking	✓	Local
Local Network Attacks		
Malicious Re-configuration	✓	✓
Malicious Re-programming	✓	7

vulnerabilities were exploitable under real deployment conditions, which could include countermeasures or other limitations (e.g., virtual fencing). Secondly, we emulated an attacker with more resources, who reverse-engineered the protocol to implement arbitrary the packet-forgery primitive, and the local-network attacks (re-configuration and re-programming).

Regarding our equipment, we used two popular SDRs devices (Nuand BladeRF x115[9] and USRP Ettus N210[10]) to develop custom receivers and transmitters with URH and GNU Radio, and Baudline[11] for visualization. We also extensively used a logic analyzer (Seleae Logic 16[12]) to capture the digital signals on the bus between the MCU and the radio chip. Note that all our attacks can be implemented on affordable hardware (e.g., less than $50) as we discuss in Section 5.

Our testing approach can be divided into following four steps.

Step 1: Reverse Engineering of the RF Communication While using the remote controller under test, we leveraged a software spectrum analyzer and a waterfall visualization to estimate the RF parameters (i.e., carrier frequency, modulation scheme, channel spacing, and symbol rate).

Given the lack of reliable documentation, and manuals that were often vague and inaccurate, we proceeded in two ways. First, we tried to identify the macro family of modulation schemes (e.g., FSK, ASK, MSK, OOK) by visualizing the signal captures in the time domain—with Baudline or URH. On the waveform, we measured the minimum distance (in milliseconds) between two subsequent modulated symbols, so to obtain the symbol rate.

Secondly, we used a logical analyzer to read the data exchanged between the MCU and the radio chip, confirm the RF parameters, and resolve ambiguities. In the best case (e.g., UART), this data was simply the digital signal as sent

[9] `https://www.nuand.com/product/bladerf-x115/`
[10] `https://www.ettus.com/product/details/UN210-KIT`
[11] `http://www.baudline.com/`
[12] `https://www.saleae.com/`

"as is" to the radio for modulation. In more complex cases we needed ad-hoc protocol-decoding work, because the data line was actually the *transport* for a custom protocol used by MCU and radio chip to communicate.

Step 2: Payload Parsing At this point, we focused on the data past the preamble and sync words. We proceeded by capturing payloads for each unique command (e.g., 'up', 'down', 'left', 'right'), to build a large corpus for statistical analysis (e.g., calculate the frequency of each value in each position of the payload to spot field separators and other patterns). Using techniques like differential analysis, we reverse engineered the application layer to understand how the payload encoded each command.

Step 3: RF Attacks Implementation We began by confirming the replay attack primitive. We captured and replayed the signal corresponding to each command while checking that the correct relay switches triggered. Using the information obtained from the previous two steps, we developed an ad-hoc SDR transmitter to synthesize arbitrary packets, and confirmed whether they were correctly interpreted by the RX.

Step 4: Local Network Attack Implementation We verified the presence of vulnerabilities that would allow malicious re-programming (i.e., through the configuration interfaces). With this attack, an adversary who has taken control of the endpoint connected to the remote controller would be able to reprogram the controller with malicious firmware and implement sophisticated attacks (e.g., backdoors in the original code). Even without replacing the original firmware, the attacker could still set the configuration parameters in a dangerous way (e.g., pressure of the 'up' button would trigger the opposite action).

4.1 In-lab Testing of Vendor 1 (SAGA)

For this evaluation we purchased a SAGA1-L8B controller, which consists of a TX with 6 single-step push buttons plus e-stop, and its RX[13]. By opening the enclosures we identified the MCU (TI MSP430F11) and the radio chip (Infineon TDA5101).

Step 1: Reverse Engineering of the RF Communication After checking the manual and the available FCC documentation, we used the spectrum analyzer to confirm the carrier frequency (i.e., 438,170 MHz).

We could clearly see the characteristic shape of a FSK modulation, with $n = 2$ deviations around the carrier, and a 8,333 bps symbol rate. This finding was corroborated by a schematic obtained from the FCC website, which showed a bus line labeled "FSKDATA" between the MCU and the radio chip. This was a good hint that the line was used to send the digital signal straight to the radio chip for modulation. Indeed, by tapping into this bus with the logical analyzer we were able to see the square wave and confirm the RF parameters observed on the spectrum.

[13] `http://www.sagaradio.com.tw/SAGA1-L6B.html`

Custom SDR Receiver Then, we used URH to develop a custom SDR receiver to decode the communication. Upon looking at several captured packets (by pressing all the buttons in various combinations), we were able to isolate the preamble and the sync words, which were the same in all packets.

With a simple guess we learned that the packets were encoded with Manchester, a well-known case of binary phase-shift keying where the data controls the phase of a square wave carrier, which frequency is the data rate.

Step 2: Payload Parsing Surprisingly, the payload was transmitted in clear with neither obfuscation nor encryption. Moreover, each command (e.g., 'right') would always generate the very same packet and signal. This is a very poor design choice. We further confirmed this finding by purchasing a second controller kit to compare the payloads from the same commands.

We used our SDR receiver to collect several payload samples, which all exhibited the same structure (shown in Figure 4). Worse, the checksum was a fixed value: Another poor design choice. Finally, by reverse engineering the firmware that we extracted from the MCU (see **Step 4**), we obtained further confirmation that the payload structure was correct.

Step 3: RF Attacks Implementation We first confirmed the replay and e-stop abuse attacks. As we had fully reverse-engineered the packet structure, we were also able to forge arbitrary, valid command packets, which the RX interpreted correctly.

Then, we tested pairing hijacking. To this end, we executed the pairing procedure as described in the manual: We pressed a special sequence of buttons on the TX already paired, which in turn sent the pairing packets via radio. In the case of this vendor, the procedure must be initiated *within the first 4 minutes* upon the RX is powered on; by design, the RX automatically accept pairing commands within this time frame. We first verified that the pairing hijacking attack could be carried out with a reply primitive, which worked out of the box. As a bonus for the attacker, the TX already paired would lost the pairing. Having reverse engineered the paring packet, we were also able to forge arbitrary pairing packets, with any codes and arbitrary target device. By exploiting this vulnerability an attacker would be able to automatically pair with multiple RXs within range, with neither local access nor user intervention.

Step 4: Local Network Attack Implementation We first confirmed that no software or hardware protection prevented unattended re-configuration.

Then, we verified if any protection would prevent unattended re-flashing of the MCU memory. Using the Bus Pirate[14], we collected the data exchanged during a (legitimate) flashing procedure.

We noticed that the client software sent the BSL password [11] in clear text. We intercepted it and used it to read the content of the flash memory and dump

[14] `http://dangerousprototypes.com/docs/Bus_Pirate`

the firmware[15]. Since the mass-erase protection was disabled, we were even able to test the BSL password that we intercepted without the risk of mass erasure (i.e., in case of incorrect password).

Upon disassembling the firmware with IDA Pro, we found that the code-integrity check was based on a simple "secret" statically stored in the code, making such integrity check easy to bypass. This confirmed that it was possible to craft a custom (malicious) firmware on this controller.

4.2 In-lab Testing of Vendor 2 (Juuko)

We purchased a Juuko JK800 transmitter (8 buttons plus e-stop and start) with its RX, plus the USB programming dongle and configuration software[16]. After removing the conformal coating that covered the text printed on each chip's package, we found out that this radio kit consists of two MCUs (Microchip PIC16L and PIC18L) and radio daughter board based on the TI CC1120.

Step 1: Reverse Engineering of the RF Communication We first identified the carrier frequency (434.192 MHz) via spectrum analysis.

Regarding the modulation, the manual mentioned GFSK[17], with no further details. The waveform showed 4 frequency deviations ($\pm 1d, \pm\frac{1}{3}d$, where d indicates the deviation's offset) implying a 4-ary alphabet (i.e., `00`, `01`, `10`, `11`). However, the waveform showed 2 deviations ($1d$ and $-1d$) in the first half the packet, and 4 deviations in the rest of the packet. Interestingly, a careful read of the CC1120 data sheet revealed that the radio used a 2-ary alphabet for preamble and sync words, and a 4-ary alphabet for the rest. Although we easily calculated the symbol rate on the waveform with URH, figuring out the symbol-deviation mapping was challenging (e.g., was `00` $\mapsto 1d$, $\frac{1}{3}d$, or what else?).

To solve this ambiguity we tapped into the serial pins of the CC1120, in the hope of deriving some useful information. Unlike the SAGA device, which used the serial line to send the digital signal to the radio, this Juuko device used the serial peripheral interface (SPI) as a transport for a non-standard protocol, allegedly developed by Texas Instruments. Such custom protocol was used by the radio driver to configure the RF parameters (e.g., carrier frequency, deviation, modulation), as well as to fill the CC1120 buffers with the data to be transmitted. This and similar SPI bus protocols abstract registry-access operations into a simple sets of transactions (e.g., read, write, transmit, receive). Thus, from a reverse-engineering perspective, it is very important to decode these custom protocols, to understand how the MCU is using the radio. However, there are no such tools like Wireshark, because such protocols vary across vendors. Therefore, using the specifications from the data sheet [13], we implemented a tool that

[15] Write-only operations are normally permitted even without password, but only limited to the code area (i.e., not the boot loader). These are not very useful, because one could blindly write data into the flash.

[16] `http://www.juukoremotecontrol.com/en/products/transmitter/jk-800-en`

[17] A FSK variant in which a Gaussian filter is applied to the signal to smoothen level transitions

decodes the byte streams obtained from the SPI into meaningful transactions. With this tool we retrieved all RF parameters and the transactions sent from the MCU to the radio.

Custom SDR Receiver At this point, we implemented a GNU Radio receiver that supported the transition from 2-FSK to 4-FSK after the sync words. Upon signal pre-filtering, the custom SDR-based receiver uses the quadrature-demodulation GNU Radio block to obtain the waveform in the time domain. On this, we used a generic 4-FSK block to derive the bit stream. Relying on the CC1120 data sheet and the register settings decoded from the SPI transactions, we knew which frequency deviations where assigned to which symbols. We wrote a custom GNU Radio block to interpret the first 8 bytes (preamble and sync words) of each packet as 2-FSK, and the rest as 4-FSK. To understand the mapping between 2-FSK and 4-FSK, refer to Figure 3: $-1d$ is either `01` (in 4-FSK) or `0` (in 2-FSK) and $+1d$ is either `11` (in 4-FSK) or `1` (in 2-FSK).

To confirm that our receiver was correct, we checked that the payload obtained from the SPI transactions matched the payload obtained from the receiver.

Step 2: Payload Parsing Looking at the decoded SPI transactions, we understood that the CC1120 was set to use the packet format specified in [13, page 45]—and depicted in Figure 2. However, although the packet format was known, the *payload* (application layer) contained further structured data, which we needed to decode.

The payload looked obfuscated, and subsequent pressures of the same button would result in distinct payload values, although exhibiting a repeating pattern. We automated the procedure of collecting all of these "variants" by using a software-driven switch attached to each physical button. Thus we created a script to "push" a button, sniff the RF and SPI traffic, "release" the button—for all buttons.

By analyzing the payloads we collected, we noticed that the first byte was taking all the 256 values from `0x00` to `0xFF`. So we hypothesized that such byte would be a sort of obfuscation seed, which was unknown to us. Short of options, we performed differential analysis by taking into account the paring code (4 bytes), which we expected to show in the payload or be used somehow to derive parts of it. Using the USB programming dongle and configuration software, we set both TX and RX to use `0x00000000` as the pairing code. Then,

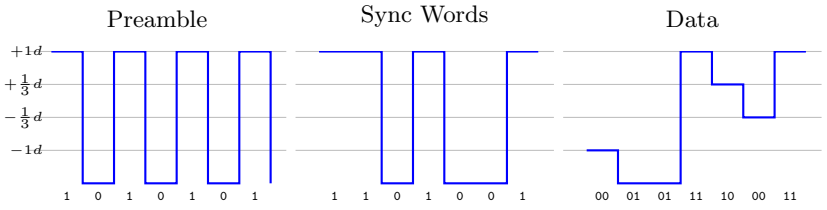

Figure 3. Data modulation with mixed 2- and 4-ary alphabet using FSK.

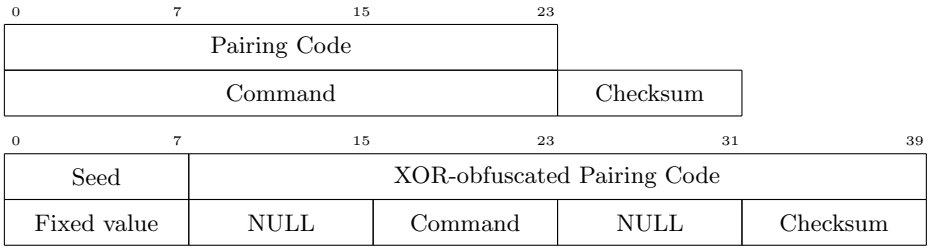

Figure 4. Payload structure for SAGA (top) and Juuko (bottom) controllers.

we collected 1 minute worth of payloads for each button. Following our hypothesis that the first byte was a seed, we compared the two datasets of payloads side by side, using such first byte as the join key. At this point, we could clearly see repeating values, such that we could guess the position of the encoded command (heartbeat, movement, start, e-stop), and the checksum. Also, we noticed that 4 bytes (exactly the length of the pairing code!) exhibited a "shifting" pattern. We exhaustively tried all binary operators until we found out that those 4, shifting bytes encoded the original (secret) fixed pairing code, in a XOR-obfuscated form. In other words, we reconstructed the obfuscation function by enumeration. At this point, an attacker would need to purchase a device similar to the target one, reset its pairing code to zero to construct such obfuscation function, and used it to de-obfuscate the secret pairing code of any target device. With that, an attacker has full packet-forging capabilities, following the structure reported in Figure 4 (bottom).

We could speculate that the seed byte could serve as a rolling code. However, while the TX kept sending unique packets, the RX would not enforce any synchronization based on the seed value. This small step would have increased the barrier for the attacker.

Step 3: RF Attack Implementation We used URH to confirm the replay attack primitive and the e-stop abuse attack, which worked out of the box.

With the knowledge we obtained from reverse engineering the radio protocol, we extended our GNU Radio receiver into a comprehensive transmitter able to intercept, modify or synthesize any transmission. We used this to confirm the arbitrary packet-forgery attack.

Then, we tested the pairing-hijacking attack, which worked out of the box with both replayed and forged packets. However, unlike the SAGA, the Juuko design is more secure because it requires pressure of a *physical* button (on the RX) in order to start the pairing procedure. This changes the attacker model completely, because it would require an attacker to be on the victim's premises and in some cases even open the receiver's enclosure to reach the button. Thus, despite the radio protocol is insecure (because of the lack of rolling code during the pairing), the physical countermeasure makes this attack much less practical.

Figure 5. Example of on-site testing in manufacturing plant.

Step 4: Local Network Attack Implementation Using the software provided by the manufacturer, we confirmed that no authentication is required to perform full re-configuration. As a result, an attacker resident on the endpoint would be able to maliciously re-configure all commands on both RX and TX.

We were not able to confirm full re-programming capabilities. By looking at the traffic on the bus lines with a logical analyzer, we understood that one MCUs implemented the application protocol and the other one managed firmware loading and updating through USB and in-circuit programming port. Using the in-circuit programming port and Microchip's MPLAB X IDE, we verified that code-read protection was enabled on the flash memory, effectively preventing casual attackers from obtaining the firmware, or changing the boot loader.

4.3 On-site Testing

To confirm the in-lab findings, include more vendors, and focus on real-world deployments, we started by visiting various facilities among factories, transportation hubs, and construction sites. We conducted tests after working hours and only upon approval by the facility management, in full compliance with safety regulations (e.g., wearing helmets). Overall, our experience was very positive, and facility managers and operators were supportive and curious to know that our research could reveal potential risks and improve the technology.

We first scouted an area in the north of Italy for candidate sites. Within 2 days, we found 24 facilities, of which 11 gave us permission: 2 manufacturing plants, 8 construction sites, and 1 transportation hub, providing a good variety of machinery, operated with remote controllers from 7 vendors (Table 2).

In the following weeks we visited the selected facilities, armed with two BladeRF devices and two laptops, as exemplified in Figure 5. Following a black-box approach, we played the role of an attacker interested in controlling (via the replay-attack primitive) and disrupting (via e-stop abuse) the target machines.

Table 2. Summary of the on-site evaluation (all vendors distribute worldwide).

Vendor	H.Q.	#Deployments	Replay Attack	E-Stop Abuse
Circuit Design	Japan	1	✓	N/A
Autec	Italy	5	✓	✓
Hetronic	US	3	✓	✓
Elca	Italy	1	✓	✓
Telecrane	Taiwan	1	✓	✓

We operated from within the RF range, with no physical access to the any equipment and no prior knowledge about model or brand name of the controllers. We then started capturing RF signals while an operator was using the remote controller. With these captures, we confirmed that both the attacks worked out of the box, which meant that all tested devices were vulnerable, due to the lack of of security mechanisms like rolling code or authentication.

We also ran range tests, to understand from how far an average hypothetical miscreant would be able to launch the attacks. Overall, the average attack range was around 300 meters in open space.

5 Remote, Stealthy and Persistent Attacks

To show that the stealthy attacker model mentioned in Section 2 is realistic, we designed and implemented RFQuack, a *modular* embedded device as small as key fob that is powered by a battery, featuring Wi-Fi or cellular connectivity.

Also, we were motivated to create a tool that would benefit researchers beyond this project, because the choice of RF-analysis tools is limited to either pre-made hardware dongles (e.g., the YardStick One showed in Figure 6, PandwaRF) or SDRs. Hardware dongles consist of an MCU with a RF chip, and a firmware that exposes an API to change the RF parameters and transmit or receive data. Although easy to use, the capabilities are constrained to the RF chip's features, which cannot be changed. For example, these dongles make it straightforward to decode data modulated with 2-FSK (ASK and other schemes), but 4-FSK is simply not supported by the chip, which is irreplaceable as soldered on the PCB.

Despite their flexibility, SDRs are slower than hardware radios and, most importantly, require domain-specific programming work and signal-processing knowledge. For instance, creating a software receiver for the TI CC1120 4-FSK was not trivial, even for seasoned computer security researchers.

RFQuack instead supports multiple, *interchangeable* radio modules and exposes the same simple API. Thus, to target a CC1120 radio, we would just need to plug a CC1120 module. Differently than existing choices, the firmware of RFQuack is Arduino based, so it is easier to adapt to a wide variety of tasks.

For example, we used RFQuack to implement the packet-forging primitive and the e-stop attack by relying only on the API and no need to write any code. Indeed, RFQuack exposes a packet-manipulation API that allows the user to

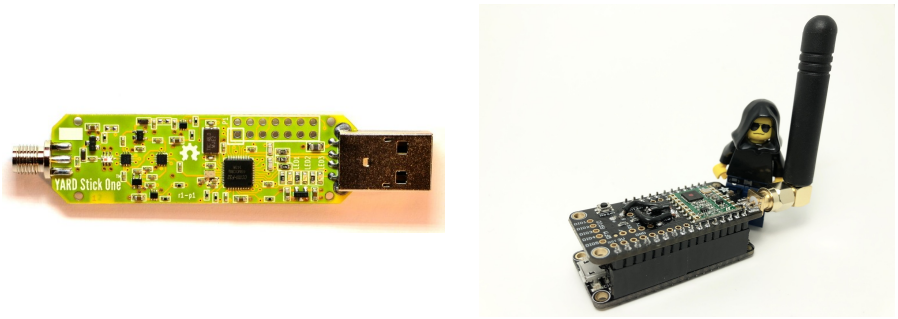

Figure 6. YardStick One RF-hacking dongle (left) and RFQuack (right), a modular research and attack tool.

specify a set of per-packet modifications with byte-level granularity (e.g, alter the 2nd byte and substitute its value with Y, or XOR the 4th byte with value Z), along with customizable regular expression rules. For example, RFQuack can be set to wait for a packet that matches a given regular expression (e.g., fingerprint of the target model and vendor), set two bytes with given values, and re-transmit the resulting packet (e.g., so obtaining an e-stop or start command).

6 Responsible Disclosure and Recommendations

Industrial remote controllers are an unexplored domain for security research. This gave us the opportunity to uncover and learn about unknown, important problems, and provide recommendations. As we believe that one of the objectives of security research is to improve the current situation (e.g., by working with vendors, associations, or standardization bodies), we hereby share the lesson learned during the challenging disclosure process.

Responsible Disclosure Process Overall, we identified and reported 10 vulnerabilities[18] affecting products of 7 vendors. We began the disclosure process by reaching out to the vendors through the Zero Day Initiative (ZDI), with all details of our findings, including SDR captures and working proof-of-concept exploits. We allowed up to 6 months for some vendors to respond, despite the typical disclosure policy would allow 90–120 days [25]. Indeed, some vendors did not have security under their radar, resulting in insecure design and lack of security testing in their development life cycle. We worked closely and assisted such vendors in releasing firmware updates, which, in some cases, were their very first security patches. This was not an easy process, but we believe that our work has set the ground for proper countermeasures, and helped raising awareness in this domain.

[18] CVE-2018-19023, ZDI-CAN-6183 [1], ZDI-18-1336, ZDI-CAN-6185 [1], ZDI-18-1362, ZDI-CAN-6187 [1], CVE-2018-17903, CVE-2018-17921, CVE-2018-17923, CVE-2018-17935.

Security Recommendations In the light of our findings, we encourage vendors to adopt well-known standard protocols (some vendors responded positively to this recommendation). When ad-hoc protocols must be used, rolling-code mechanisms and encryption should be used to raise the bar for attackers. Modern radio transceivers even have hardware support for encryption (e.g., CC1110Fx/CC1111Fx [19]), making it easier to implement (yet harder to deploy than a software patch). Nevertheless, with limited bandwidth, a rolling-code implementation would not be robust to active attacks such as RollJam [14]. Last, the firmware should be protected against physical access (e.g., using tamper-proof enclosures) and reverse engineering.

Users should opt for controllers that offer dual-technology devices (e.g., with virtual fencing) and open, and well-known RF standards. Users and system integrators should install the firmware updates made available by the vendors in response to our disclosure. When fixed-code protocols are the only option, such pairing code should be changed periodically.

Although all the vulnerabilities we discovered can be patched in software, patch deployment is far from trivial, mostly for practical reasons. Moreover, these systems can have decades-long life time, increasing the exposure to attacks.

Standardization Bodies Industrial radio remote controllers adhere to safety standards, which at the moment consider only failures affecting (and happening in) the physical world. During our research, we have been informed that our work is currently being used to motivate working groups of European-level standardization bodies to introduce new requirements in the standards (e.g., immunity to replay attacks), because the threat model must include attackers actively trying to "interfere" with the RF communication.

7 Related Work

Our work looks at RF communication security *in the context* of OT environments.

Since industrial remote controllers rely on custom protocols, research in this area is scarcer than on standard protocols. The latter are however not free from vulnerabilities. For example, Balduzzi et al. [3] analyzed and uncovered vulnerabilities in the protocol used for vessel positioning and management. Kerns et al. [15] discussed theoretical and practical attacks against the GPS systems employed in unmanned aerial vehicles, while Bhatti et al. [4] explored the problem of the use of GPS in the marine traffic and vessels. Costin et at. [6] highlighted security vulnerabilities, including spoofing, in ADS-B, a well-known protocol use in the aviation sector. Francillon et al. [10] described replay attacks against passive keyless entry and start systems of modern cars. Wright [23] proposed an offensive security research framework for the ZigBee protocol, and Vidgren et al. [21] proposed two attacks to sabotage ZigBee-enabled devices. Fouladi et al. [9] discovered a vulnerability in the AES implementation in a Z-Wave lock. Interestingly, these consumer-grade technologies offer a greater security assurance than industrial remote controllers.

Although OT security researches may have touched topics related to our work, none of them looked into the domain of industrial remote controllers or RF technology for industrial applications. Quarta et al. [18] looked at industrial robots. Although some industrial robots can be operated via RF technology, [18] focused exclusively on TCP/IP-based protocols. Fleury et al. [8] presented a taxonomy of cyber attacks against SCADA-based systems employed in the energy sector. More generally, Papp et al. [16] conducted a systematic review of the existing threats and vulnerabilities in embedded systems. The authors touched the problem of Internet-exposed industrial routers and critical industrial facilities running vulnerable or mis-configured services.

8 Conclusions

We analyzed how industrial machinery is controlled and operated via RF technology. Despite marketed as "industry-grade systems" and used for safety-critical applications, the RF protocols in use are less secure than their consumer-grade equivalent (e.g., garage- or car-door openers).

Our findings confirm that these systems are designed with no or little security in mind. The most striking aspect is that the vulnerabilities we found are not *implementation* flaws (e.g., buffer overflows): Including a rolling-code mechanism is a *design* choice.

References

1. Andersson, J., Balduzzi, M., Hilt, S., Lin, P., Maggi, F., Urano, A., Vosseler, R.: A security analysis of radio remote controllers for industrial applications. Tech. rep., Trend Micro, Inc. (01 2019), https://documents.trendmicro.com/assets/white_papers/wp-a-security-analysis-of-radio-remote-controllers.pdf
2. Arkansas: Heavy load accident (2013), https://cdn.allthingsnuclear.org/wp-content/uploads/2015/02/FS-181-PDF-File-with-links.pdf
3. Balduzzi, M., Pasta, A., Wilhoit, K.: A security evaluation of AIS automated identification system. In: Proceedings of the 30th Annual Computer Security Applications Conference, ACSAC 2014, New Orleans, LA, USA, December 8-12, 2014. pp. 436–445 (2014). https://doi.org/10.1145/2664243.2664257, http://doi.acm.org/10.1145/2664243.2664257
4. Bhatti, J., Humphreys, T.E.: Hostile control of ships via false gps signals: Demonstration and detection. NAVIGATION: Journal of the Institute of Navigation **64**(1), 51–66 (2017)
5. Blossom, E.: Gnu radio: Tools for exploring the radio frequency spectrum. Linux Journal **2004**(122), 4– (June 2004)
6. Costin, A., Francillon, A.: Ghost in the air (traffic): On insecurity of ads-b protocol and practical attacks on ads-b devices. Black Hat USA pp. 1–12 (2012)
7. CYREN: Cyber pirates targeting logistics and transportation companies (2018), https://www.cyren.com/blog/articles/cyber-pirates-targeting-logistics-and-transportation-companies

8. Fleury, T., Khurana, H., Welch, V.: Towards a taxonomy of attacks against energy control systems. In: International Conference on Critical Infrastructure Protection. pp. 71–85. Springer (2008)
9. Fouladi, B., Ghanoun, S.: Security evaluation of the z-wave wireless protocol. Black hat USA **24**, 1–2 (2013)
10. Francillon, A., Danev, B., Capkun, S.: Relay attacks on passive keyless entry and start systems in modern cars. In: Proceedings of the Network and Distributed System Security Symposium (NDSS). Eidgenössische Technische Hochschule Zürich, Department of Computer Science (2011)
11. Goodspeed, T.: Practical attacks against the msp430 bsl. In: Twenty-Fifth Chaos Communications Congress (2008)
12. Greenberg, A.: Crash override malware took down ukraine's power grid last december (2017), https://www.wired.com/story/crash-override-malware/
13. Instruments, T.: Cc1120 user's guide (2013), http://www.ti.com/lit/ug/swru295e/swru295e.pdf
14. Kamkar, S.: Drive it like you hacked it: New attacks and tools to wirelessly steal cars (2015), https://samy.pl/defcon2015/
15. Kerns, A.J., Shepard, D.P., Bhatti, J.A., Humphreys, T.E.: Unmanned aircraft capture and control via gps spoofing. Journal of Field Robotics **31**(4), 617–636 (2014)
16. Papp, D., Ma, Z., Buttyan, L.: Embedded systems security: Threats, vulnerabilities, and attack taxonomy. In: Privacy, Security and Trust (PST), 2015 13th Annual Conference on. pp. 145–152. IEEE (2015)
17. Pohl, J., Noack, A.: Universal radio hacker: A suite for analyzing and attacking stateful wireless protocols. In: 12th USENIX Workshop on Offensive Technologies (WOOT 18). USENIX Association, Baltimore, MD (2018), https://www.usenix.org/conference/woot18/presentation/pohl
18. Quarta, D., Pogliani, M., Polino, M., Maggi, F., Zanchettin, A.M., Zanero, S.: An experimental security analysis of an industrial robot controller. In: 2017 IEEE Symposium on Security and Privacy (SP). pp. 268–286 (May 2017). https://doi.org/10.1109/SP.2017.20
19. Texas-Instrument: Cc1110fx / cc1111fx, http://www.ti.com/lit/ds/symlink/cc1110-cc1111.pdf
20. TrendMicro: Triton wielding its trident – new malware tampering with industrial safety systems (December 2017), https://www.trendmicro.com/vinfo/us/security/news/cyber-attacks/triton-wielding-its-trident-new-malware-tampering-with-industrial-safety-systems
21. Vidgren, N., Haataja, K., Patino-Andres, J.L., Ramirez-Sanchis, J.J., Toivanen, P.: Security threats in zigbee-enabled systems: vulnerability evaluation, practical experiments, countermeasures, and lessons learned. In: System sciences (HICSS), 2013 46th Hawaii international conference on. pp. 5132–5138. IEEE (2013)
22. Wilhoit, K.: Killdisk and blackenergy are not just energy sector threats (February 2016), https://blog.trendmicro.com/trendlabs-security-intelligence/killdisk-and-blackenergy-are-not-just-energy-sector-threats/
23. Wright, J.: Killerbee: Practical zigbee exploitation framework or" wireless hacking and the kinetic world" (2018)
24. Yaneza, J.: 64-bit version of havex spotted (December 2014), https://blog.trendmicro.com/trendlabs-security-intelligence/64-bit-version-of-havex-spotted/
25. ZDI: Disclosure policy, https://www.zerodayinitiative.com/advisories/disclosure_policy/