SCADA Radio Blackbox Reverse Engineering

Jean-Benoit Larouche[1](✉), Sébastien Roy[1], Frédéric Mailhot[1], Pierre-Martin Tardif[2], and Marc Frappier[3]

[1] Electrical and Computer Engineering Department, Université de Sherbrooke, Sherbrooke, QC, Canada
djibylarouche@hotmail.com
[2] Management School, Université de Sherbrooke, Sherbrooke, QC, Canada
[3] Computer Science Department, Université de Sherbrooke, Sherbrooke, QC, Canada

Abstract. Supervisory control and data acquisition (SCADA) systems were designed to be open, robust, and easy to operate and repair, but not necessarily secure. In recent years, there have been multiple successful attacks targeting SCADA systems. Most of them have been caused by deploying malware on a supervisory computer in order to control and manage programmable logic controllers (PLCs) and remote terminal units (RTUs). This work investigates a different potential way to control PLCs or RTUs in a plant which consists in inflitrating over-the-air (OTA) links based on SCADA wireless modems. Indeed, PLCs and RTUs are often linked to a supervisory computer wirelessly using outdated radios, with low security at the physical-layer level. An example of such a radio is the CalAmp Guardian-400 wireless modem. A blackbox reverse engineering of the physical layer of the latter is performed, which leads to complete signal demodulation and decoding. Our results demonstrate that any electronic equipment connected serially to the radio is vulnerable to wireless packet injection.

Keywords: SCADA · software defined radio · scrambling · NRZI · digital phase locked loops · FSK

1 Introduction

Radio waves can transfer information between two or more points quite effectively, over large distances, at the speed of light. However, anyone with the adequate equipment can act as a valid receiver and intercept those electromagnetic waves, or interfere with the signal by transmitting on the same time/frequency resource, commonly known as *jamming*. With the ubiquitous presence of radios in many essential aspects of our day-to-day life (such as Wi-Fi, cellphones, cars, computers, TVs, etc.), wireless security is of utmost importance. The same degree of care should be applied to any wireless device required in industrial processes and critical infrastructures underlying our modern society.

SCADA stands for Supervisory Control and Data Acquisition. A SCADA system is a combination of hardware and software that enables the automation of industrial processes by capturing Operational Technology (OT) data. Hardware such as Remote Terminal Units (RTUs) and Programmable Logic Controllers (PLCs) serve as local collection points for acquiring this OT data. This information can be acted upon directly using programmed logic or gathered by a computer commonly known as a gateway.

Gateways can come in various forms such as edge computers, Human Machine Interfaces (HMI) or a central server. One advantage of such an architecture resides in the ability to monitor and control systems from multiple locations. In the past, it has often been common practice to air gap such control system networks as they typically didn't need to interact with other corporate systems or the Internet. In theory, having this disconnection has been a sufficient security measure, however this is often no longer operationally feasible in today's connected world.

IBM Managed Security Services (MSS) data reveals there has been a 110% increase in attacks on industrial control systems since 2016 - a threat landscape that is predicted to grow at a phenomenal rate in the upcoming years [7]. A second report from Raytheon [8] mentions that 80% of companies expect an increase in cyber risk over the coming years. This is to be expected since OT systems constitute easy targets. Communication protocols designed without stringent security measures (ModBus, DNP3 [2]), corporate environments running outdated software and default passwords on embedded accounts and/or personal devices are all common security vulnerabilities which can cause a lot of damage on many different fronts (loss of operations and revenue, infrastructure shutdown, loss of physical well-being, etc.). Additionally, updating and/or replacing industrial devices can prove to be quite expensive and complicated. This directly leads to a large quantity of obsolete legacy equipment still in operation and involved in critical tasks. The present paper exposes a cyber risk linked to the usage of outdated hardware designed without stringent security measures, in an industrial power plant. More specifically, the vulnerability resides in the radios used to establish wireless links between PLCs and supervisory computers.

The popular malware attacks on SCADA systems drove a lot of research in the field of cybersecurity, but most of it focuses on potential network breaches or possible software exploits. There is very little research on the potential risks involved when using legacy RF equipment with an unsecured physical layer. The only relevant work found on this topic is presented in [4]. Therein, the reverse-engineering of a widely deployed GE MDS-9710 radio is performed. The physical layer of the latter frequency-modulates a scrambled, duo-binary coded signal as a means to transfer information OTA. Using GNU radio, successful demodulation of the MDS-9710 radio signals was achieved, thanks to their access to the theory of operation document (filed with the FCC by the manufacturer) and the digital signal processor (DSP) firmware. In a similar fashion, our work presents a step-by-step blackbox reverse engineering of the Guardian-400 wireless modem physical layer, using only an ADALM-PLUTO SDR [9] platform and

MATLAB software. No firmware access or technical document was available, besides the user manual. Still, even with very limited knowledge, our work shows how an unsecured physical layer can be simple to analyze, using widely accessible and inexpensive tools.

The CalAmp Guardian-400 commercial radio is conveniently interoperable with multiple discontinued radios such as the CalAmp DL-3400 analog transceiver, the DL-3282 analog radio, and the T-96SR wireless modem. The latter has the most interesting mode of operation since it is capable of telemetry transmission at a rate of 9600 bps and thus, is more relevant in the context of today's SCADA systems.

The T-96SR radio was developed many years ago, circa 1999 [5], when wireless data transmission was very niche. Thus, little effort was made to protect the equipment from potential wireless attacks and as a consequence, the radio consists of a simple and unsecured physical layer. From [1], it is found that the T-96SR physical layer implements differential raised-cosine minimum shift keying (DRCMSK) modulation. Before modulation, the payload bits are scrambled using a 7-bits scrambler, then differentially encoded using NRZI (Non-Return-to-Zero Inverted). No forward error correction (FEC) and packet structure is defined. Thus, it is assumed that the Guardian-400 radio behaves in the same manner when in compatibility mode with the T-96SR. This constitutes the starting point of our reverse-engineering process.

2 Previous Results

This paper is based on an in-depth signal analysis of OTA recordings, obtained by standing at a 1 km radius from a power plant with an ADALM-PLUTO SDR platform. In order to extract the transmitted bits, an FSK demodulator is required. The block diagram in Fig. 1 shows the developed FSK demodulator.

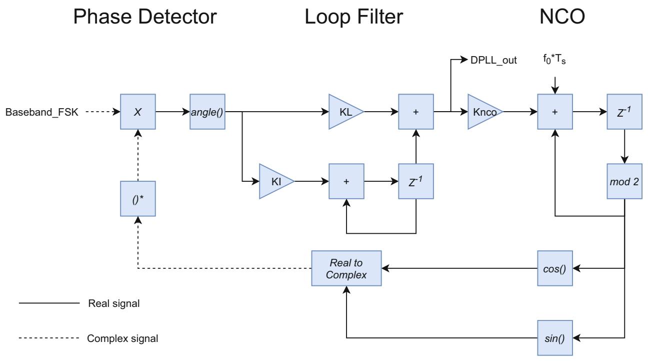

Fig. 1. FSK demodulator block diagram.

The demodulated bits are extracted from the *DPLL_out* signal. This signal follows the ADALM-PLUTO SDR sampling rate of 528 KHz and is affected by multiple RF impairments. In order to extract the bits, a bit extractor, based on the digital phase-locked loop (DPLL) presented in Fig. 1, is used. The phase detector is replaced by an inductive minimum mean-square error (MMSE) timing error estimator, which samples the *DPLL_out* input at twice the expected symbol rate of 9600 bps, then calculates the discrete-time derivative. Figure 2 illustrates this concept.

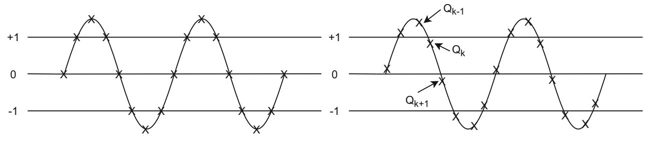

Fig. 2. Correct sampling phase vs. late sampling phase at twice the symbol rate.

The timing error estimation is calculated using the following formula:

$$\tau_{k+1} = \tau_k - (Q_k - A_k)(Q_{k+1} - Q_{k-1})$$

where $A_k = \pm 1$, following the polarity of Q_k. A NCO (Numerically Controlled Oscillator) is used to generate the sampling instants 19200 Hz, with its phase being controlled by the filtered timing error estimate. The complete block diagram, including the MMSE timing error estimator, a loop filter and the NCO is illustrated in Fig. 3.

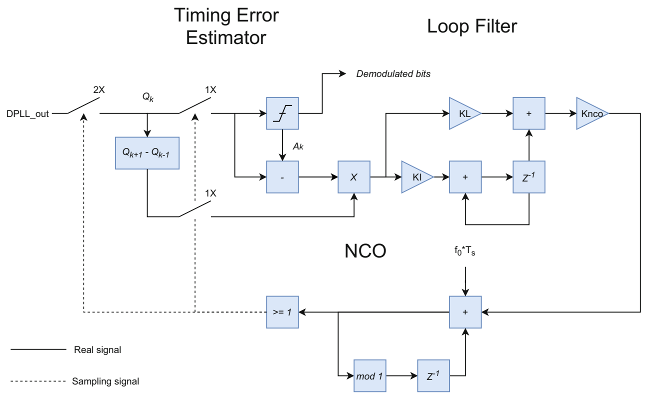

Fig. 3. Symbol extractor block diagram.

The first phase concludes with the fact that by using a signal processing software such as MATLAB, one can use DPLLs to demodulate a Guardian-400 signal down to its transmitted bits. These bits however, are assumed to be encoded and scrambled at the physical layer and thus, no direct extraction of payload bits can be performed.

The present paper goes deeper into the physical layer analysis of the radio, thanks to its commercial availability. It is shown that one could potentially recover transmitted payload bits from OTA recordings but more importantly, could inject its own message at the output of the radio's serial port. Thus, any piece of software or hardware connected to the Guardian-400 is exposed to potential security threats.

2.1 Hardware Description

The two main pieces of hardware and their relevant settings are presented in Table 1.

Table 1. Hardware configuration

	GUARDIAN-400	*ADALM-PLUTO SDR*
Tx frequency (MHz)	450	425.5
Rx frequency (MHz)	425.5	450
Data rate (bps)	9600	9600
Bandwidth (kHz)	25	25
Sampling rate (kHz)	N.A	528

Both devices can be connected to a Windows 10 laptop using USB cables. The Guardian-400's software is used to configure and communicate with the Guardian-400 radio through a USB-to-serial cable and MATLAB is used to configure and communicate with the ADALM-PLUTO SDR.

3 Tests Description

3.1 Loopback Test

As mentioned in Sect. 2, one can demodulate the transmitted bits using DPLLs but not unravel their encoding, thus leaving one unknown layer to traverse to get to the payload. In order to investigate these unknowns, the following test plan was enacted:

1. Configure the Guardian-400 to transmit a known test vector;
2. Perform an OTA recording using the ADALM-PLUTO SDR;
3. Perform FSK demodulation and symbol clock recovery on the recording using DPLLs;
4. Using the recovered encoded bits, draw conclusions regarding the physical layer processing blocks.

In order to transmit a known test vector, the Guardian-400 is configured in transmit mode with a 1-second interval ASCII pattern. The pattern used to build the packets has the following format (55 ASCII characters):

000ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz
001ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz
- - -
998ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz
999ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz
000ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz
001ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz

The second step towards this goal is to perform a recording using the ADALM-PLUTO SDR. This is easily achieved in MATLAB using the ADALM-PLUTO SDR FM receiver example from Mathworks as a starting point. The real portion of a 2-seconds recording is shown in Fig. 4.

From Fig. 4, it can be observed that the Guardian seems to transmit data continuously once in transmit mode. The ASCII pattern is only 55 characters in length. Short transmit bursts were expected every second, in an otherwise unbroken silence. At this point, the actual position of the payload within this recording is unknown. Nevertheless, the DPLLs developed in the first phase can be used in order to demodulate the bits. As a reminder, the demodulated bits are (theorically) differentially coded using NRZI and scrambled using a 7-bits scrambler. The details about both processes were still unknown at this point in our investigation.

In order to ensure that our recording included the known test vector, the recording can be transmitted back to the Guardian-400 for demodulation. To do so, a 2-FSK modulator was developed. There are multiple ways to implement such a modulator, but an approach which uses the NCO in Fig. 1 seemed the most logical.

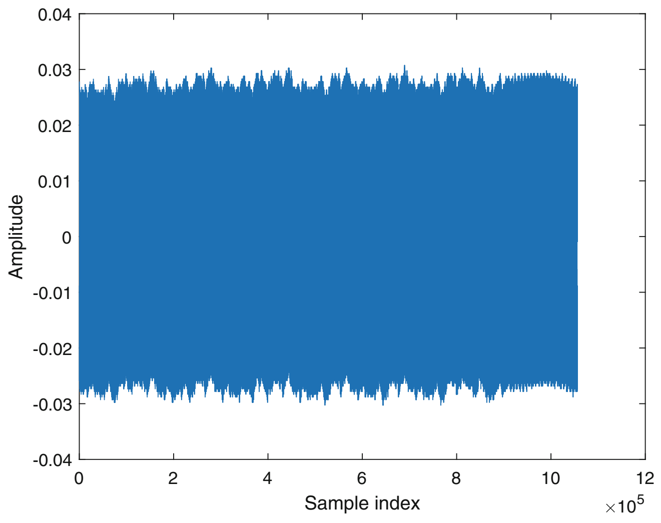

Fig. 4. Recording of the Guardian-400 ASCII pattern transmission.

Generating the 2-FSK waveform using the demodulated bits and transmitting them OTA using the ADALM-PLUTO SDR yielded the following results on the Guardian-400 software ASCII terminal.

$$\text{FGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz} \qquad (1)$$

A portion of the expected ASCII pattern was correctly received, but it is important to point out that a total of 19200 bits (2-second recording at 9600bps) were sent by the ADALM-PLUTO in order to achieve this result, which is much more than the 47 received ASCII characters in output (1) (which gives 376 bits assuming an 8-bits ASCII representation). At that point, it is obvious that the pattern of interest is somewhere in the recorded file. However, it is not clear where and it is also not clear why there is so much overhead.

3.2 Payload Analysis

Following the loopback test, it is now of interest to dig deeper into the received payload bits and try to reverse-engineer the modulation process (from ASCII characters to differentially-coded and scrambled bits). As a first step, the bits of interest need to be isolated from the overhead bits in order to perform further processing. By visual inspection of the demodulated bits in Fig. 5, one can identify surprisingly long series of 1's or 0's at many points in the bit stream.

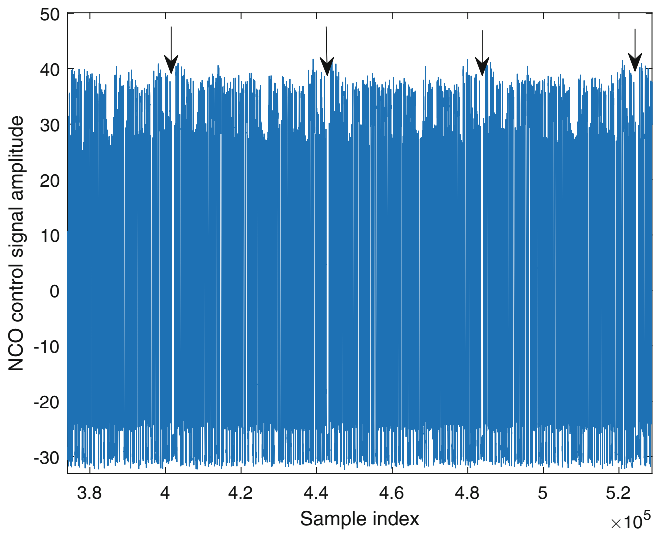

Fig. 5. Sequences of same polarity bits from the demodulated bit stream.

It is surprising since the bits are supposed to be scrambled, which should remove such scenarios. This could indicate the presence of some kind of preamble or synchronization pattern. Secondly, these series of bits also seem to appear periodically. In order to check for the periodicity, convolution is performed using the following coefficients, taken from the demodulated bits:

$$coeff = \left[1\ -1\ -1\ -1\ -1\ -1\ -1\ -1\ -1\ -1\ -1\ -1\ -1\ -1\ -1\ 1\right]$$

Plotting the result of the convolution operation gives the output shown in Fig. 6.

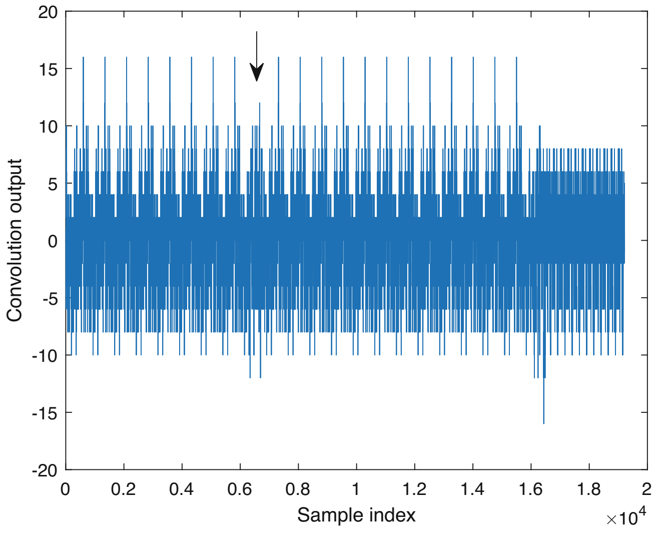

Fig. 6. Convolution output using *coeff*.

From Fig. 6, it can be observed that a good portion of the demodulated bits seems to be filled with a repetitive pattern. It also shows two other locations of interest, which are right after the first eight peaks and at the tail portion of the recording. Using the convolution peaks as delimiters, the 19200 bits can be divided into multiples blocks of bits which enables us to iteratively test for the location of our ASCII bits, by simply sending these chunks of bits, one at a time, to the Guardian-400. Using that process enables us to locate the bits of interest which are between the eight and ninth convolution peaks, inside a block of 1504 bits. This is definitely a step forward but the block still contains more than the expected 376 bits.

3.3 NRZI and Scrambler Decoding

At this point, the exact location of the payload bits is unknown. We began the analysis of the NRZI decoding and scrambling processes, which prevented us from relating the demodulated bits at the output of the DPLLs to the ASCII characters being sent. Figure 7 illustrates an initial hypothetical block diagram of the expected ASCII characters encoding/decoding process.

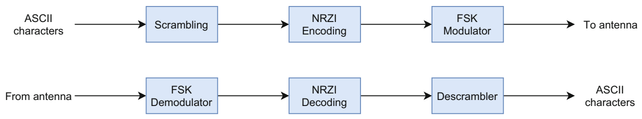

Fig. 7. Hypothetical Guardian-400 encoding/decoding process.

It is also important to mention that the block diagram in Fig. 7 acts as a starting point, extrapolated from the information in [1] and only from that source. Therefore, at this point, it is possible that additional blocks are present and/or that some information may be false, such as the length of the scrambler for example.

Compared to NRZ (Non-Return-to-Zero), NRZI is a differential encoding technique which distinguishes data bits by the presence or absence of a bit polarity transition at clock edges. Two NRZI scenarios are thus possible: a transition from one polarity to the opposite polarity could either represents a 1 or a 0.

Regarding the scrambler, a 7-bits scrambler is expected from the data in [1]. However, the architecture of the scrambler and the scrambler polynomial are unknown and thus, represent the main challenge to overcome. One popular and well known 7-bits scrambler is the 802.11 (WiFi) scrambler shown in Fig. 8.

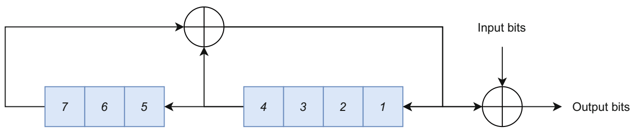

Fig. 8. 802.11 7-bits synchronous scrambler.

This is a synchronous scrambler with polynomial $1 + x^4 + x^7$. One important point to note is the fact that the input bits are not influencing the states of the registers in such architecture. As a consequence, for correct descrambling, the descrambler needs to be put into a specific state, at input bit b, usually using a preamble detection mechanism which could be possible in our case, since there are multiple overhead bits. Both the scrambler and the descrambler are identical in such an architecture.

A second possible architecture, which is simpler implementation-wise but prone to error propagation, could also be used: a self-synchronous architecture. The scrambler/descrambler of the latter is shown in Fig. 9.

These are 7-bits self-synchronous scrambler/descrambler of polynomial $1 + x^6 + x^7$. Compared to the synchronous architecture, the register states are

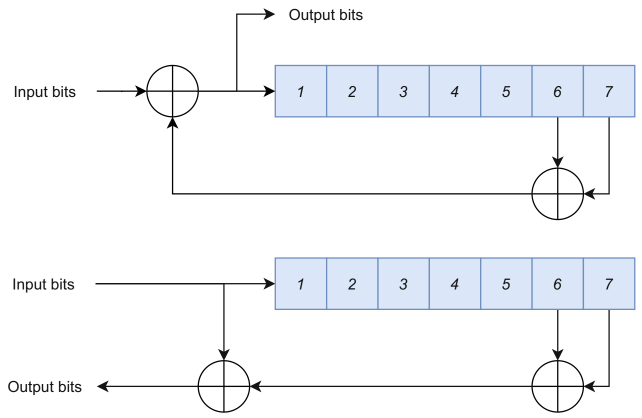

Fig. 9. 7-bits self-synchronous scrambler(top) and descrambler(bottom).

affected by the input bits and thus, a detection error at the receiver will tend to propagate through the descrambler. However, a preamble detection mechanism is not required, which reduces the required hardware resource and computing power. This architecture is definitely a possibility since there is hypothetically no packet structure defined (so no specific preamble section defined). Furthermore, its low resource usage is a good fit to the limited processing capability of an old T-96SR radio.

That being said, there remains numerous possibilities which need to be investigated. As a first investigative step, one can try to take the FSK demodulated bits from the DPLLs and decode them in MATLAB into the expected ASCII characters, using a brute-force approach, by testing all possible configurations. The 7-bits synchronous architecture is analysed first.

3.4 Synchronous Architecture

As mentioned above, synchronous scramblers/descramblers require a specific state reset mechanism for correct descrambling. In other words, at a specific bit b, the scrambler/descrambler needs to be in a specific state. This means that there is a total of four variables to test in order to cover all possible scenarios:

1. NRZI transitions either represent a 0 or a 1;
2. The starting bit b of our encoded ASCII sequence which is somewhere in the 1504 bits block;
3. The state of the descrambler at bit b;
4. The descrambler polynomial.

This looks intimidating at first but this can be coded using multiple *for* loops rather easily, it simply takes a significant amount of processing time. The output of the descrambling process is validated using a convolution operation with the 8-bits representation of the following ASCII sequence: *JKLM*.

After multiple iterations of the current approach, the expected convolution result of 32 never occurred. Descrambler sizes of 6-bits and 8-bits were also tried but didn't give the expected results.

3.5 Self-synchronous Architecture

Following the results in the previous section, the code was slightly modified in order to implement a 7-bits self-synchronous architecture. The same four variables are tested but again, the approach never gave a convolution result of 32.

4 Single Error Injection Approach

After the unsatisfactory results of the brute force approach, it is clear that some form of understanding is missing. Another approach to resolution is therefore proposed: the error injection approach. This approach consists of transmitting the 1504-bits block, which contains the ASCII characters bits, to the Guardian-400 but with the insertion of a single bit error beforehand. The goal of such a test is to study how the received ASCII characters are affected by this error and try to draw some conclusions about the unknown variables. Through trial and error, flipping the 554th bit gave the result on the Guardian-400 ASCII terminal shown below (2).

$$\text{FGH}^{\text{a}}\text{KKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz} \tag{2}$$

By careful observation, one can see that the I and J ASCII characters were affected by the single bit flip, which already gives a good hint regarding the descrambler architecture. In a synchronous architecture, a single bit error at the input gives a single bit error at the output, but it is not the case in a self-synchronous architecture. The error will affect the state of the descrambler and thus, multiple bits at the descrambler output. A closer look at the IJ ASCII characters and their error-injected version is presented in Table 2.

Table 2. I and J with error injection

I	J
01001001	01001010
^{a}	K
10101010	01001011

From Table 2, it is clear that multiple bits were affected by our single error injection. It is also clear that the 554th bit corresponds to the I character LSB (rightmost bit). The same single error injection test was repeated by flipping the 555th, 556th, 557th, 558th, 559th bits. By checking the 8-bits representation of the decoded ASCII characters, two interesting observations are obtained. First, the ASCII characters are definitely 8 bits long and second, the bits of each ASCII character are sent OTA to the Guardian-400, LSB first. This is an important point, since, in the brute-force approach, it was assumed that the *JKLM* sequence used for the convolution operation was simply the direct 8-bits representation of each character taken from the 8-bits ASCII table. So, instead of testing for the bit sequence in Table 3, one should have checked for the sequence in Table 4.

Table 3. MSB first "IJKL" bit sequence

I	*J*	*K*	*L*
01001001	01001010	01001011	01001100

Table 4. LSB first "IJKL" bit sequence

I	*J*	*K*	*L*
10010010	01010010	11010010	00110010

4.1 Multiple Bits Injection Approach

At this point, a self-synchronous scrambler is supposed, and another test comes to mind, which exploits the fact that the input bits of the descrambler can affect the register's state. The second test consists of inserting a long string of NRZI modulated ones or zeros. The following NRZI sequence is inserted in the 1504-bits block, at bit position 554:

1111 1111 1111 1111

The Guardian-400 ASCII terminal gives the result below (3).

$$\text{FGHNÿøoLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz} \tag{3}$$

New characters appear on the screen, more importantly the *ÿ* character at the expected location. The 8-bits ASCII representation of the *ÿ* character is 11111111, which tells us about the NRZI encoding. The ability to fill the descrambler registers with 1's by using an NRZI sequence which contains no polarity transition, means that a binary 1 is encoded by the absence of a transition and thus, a 0 is encoded by the presence of a polarity transition. It is now known how to communicate a 1 and a 0 to the Guardian-400 and control the state of the descrambler.

4.2 Descrambler Impulse Response

Following the results from the previous test, it is now time to test for the impulse response of the descrambler by first, feeding the descrambler with a series of 1's to put the descrambler in a known state and secondly, injecting a single 0, followed again by multiple 1's. The used NRZI coded sequence is as follows:

1111 1111 1111 1111 -1-1-1-1 -1-1-1-1 -1-1-1-1 -1-1-1-1

Transmission OTA to the Guardian-400 gives the output below (4) on the ASCII terminal.

$$\text{FGHNÿ\^{}ÿYJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz} \qquad (4)$$

The characters of interest are *ÿˆÿ* which is illuminating with regards to the number of registers of the descrambler and its polynomial. It is clear from the ASCII characters that the descrambler is in an all-ones state, then a zero is fed, which goes through all the registers, creating a sequence of multiple states before going back to the all-ones state. The 8-bits representation of the ASCII characters (left to right = MSB to LSB) is shown in Table 5.

Table 5. *ÿ* and ˆ 8-bits representation

ÿ	ˆ	*ÿ*
11111111	01011110	11111111

Simply by looking at the ˆ character from LSB to MSB, some hypothesis can be made regarding the descrambler. First and foremost, it is indeed a 7-bits descrambler and secondly, the polynomial is $1 + x^5 + x^7$, which is not an optimal choice since it is not a primitive one. One can validate these assumptions by performing the descrambling process, by hand. The proposed descrambler architecture from the all-ones initial state is shown in Fig. 10.

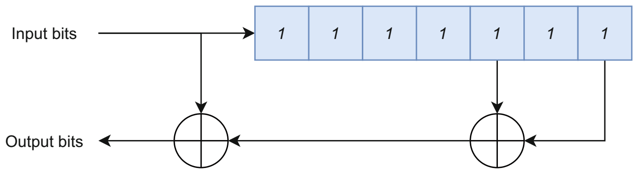

Fig. 10. Proposed self-synchronous descrambler architecture.

From Table 6, it is clear that the register length and the polynomial seem to be correct.

Table 6. Proposed descrambler impulse response

Inputs bits	Register states	Descrambled bits
0	1111111	0
1	0111111	1
1	1011111	1
1	1101111	1
1	1110111	1
1	1111011	0
1	1111101	1
1	1111110	0

4.3 Hello World! Test

With all this new information in hand regarding the encoding and decoding process, one could be tempted to send his own message to the Guardian-400, using the ADALM-PLUTO SDR. To perform this test, a modulator now needs to be developed. The modulator needs to perform the following tasks:

1. Convert the *ÿÿHello World!* ASCII message to 8-bits representation (*ÿ* is used to put the descrambler in a known state);
2. Perform a bit flip from left to right for each ASCII character, in order to feed the bits LSB first;
3. Scramble the bit sequence using a self-synchronous scrambler of polynomial $1 + x^5 + x^7$, with initial state 1111111;
4. Encode the scrambled bits differentially using NRZI, where $1 =$ No transition and $0 =$ Transition.

The test is successful if the *Hello World!* message is correctly decoded by the Guardian-400 ASCII terminal. Using MATLAB, the *Hello World!* message is encoded and sent to the FSK modulator, for transmission to the Guardian-400. The output of the ASCII terminal, by inserting our encoded bits at position 554 in the 1504 bits chunk, is shown below (5).

$$\text{FGH/ÿHello World!]JKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz} \quad (5)$$

The test is conclusive and validates our assumptions regarding the encoding and decoding process of the Guardian-400.

4.4 Updated Brute-Force Approach

Having successfully transmitted the *Hello World!* message, it could now be interesting to go back to the brute-force approach and validate if the approach was legitimate after all. Performing the brute-force approach, with the correct descrambler polynomial and architecture, with the proposed LSB first *JKLM* convolution sequence still did not yield the expected result of 32.

However, performing the same approach with another sequence, *GHIJ*, did give the expected convolution result of 32. The brute-force approach also gave some interesting information regarding the location of the sequence and the state of the descrambler. It actually gave multiple locations, which comes from the fact that the polynomial is not primitive.

From these results, the longer sequence *GHIJK* was tried but did not give the expected convolution results. Inspection of the descrambled data shows the presence of the following ASCII character, which follows the *GHIJ* sequence: ∼ or *0x7E*. This character appears periodically in the descrambled sequence (for each 3 or 4 ASCII characters) and also appears to be added to the sequence, since now, if the test sequence becomes *GHIJ∼K*, the convolution results gives the expected result.

Bottom line, going back to the brute-force approach gave an additional piece of information, the presence of a ∼ character, probably used for synchronisation purposes since it never appeared on the ASCII terminal in our tests. This is validated easily by performing the *Hello World!* test with the following message:

∼∼∼∼∼∼H∼el∼lo∼ Wo∼r∼ld∼∼∼∼∼∼!

Multiple tests were performed with multiple different numbers of sync characters and insertion locations and the result on the ASCII terminal is still simply *Hello World!*.

4.5 Preamble Detection

From the results of the previous tests, it is now easy to find and remove the excess bits from the 1504 bits chunk, simply through trial and error. But simply sending our NRZI encoded and scrambled *Hello World!* message without any of the previous bits of the 1504 bits does not work. There is definitely some kind of preamble which is used to trigger the radio's decoding process. Through trial and error, it is found within a few minutes that bits 334 to 513 seem to include that preamble. Creating an NRZI message with these bits, followed by the scrambled and encoded *∼∼∼∼∼∼H∼el∼lo∼ Wo∼r∼ld∼∼∼∼∼∼!* message, gives the output on the ASCII terminal, shown below (6).

$$\text{Hello World!} \tag{6}$$

This additional piece of information is definitely useful. This preamble can now be used to trigger the radio demodulation mechanism but this preamble could also be used in order to intercept actual OTA signals from Guardian-400/T-96SR radios and potentially extract sensitive information, since the payload bits follow the preamble.

5 Conclusion

In this report, a thorough analysis of the Guardian-400 physical layer is achieved. As presented in Fig. 7, a 7-bits scrambler and an NRZI encoder are the only processing blocks which manipulate the payload bits before FSK modulation. Additionally, the existence of a synchronisation character and a preamble is observed. In order to perform these tests, one simply needs a Guardian-400, an off-the-shelf SDR platform and a laptop. The absence of any packet structure, authentication process and forward error correction mechanism makes the Guardian-400 physical layer unsecure and easy to investigate. This work illustrates the necessity to add additional security layers to any equipment connected to a Guardian-400. This recommendation can confidently be extended to any SCADA system which includes radios which are interoperable with the latter, such as the T-96SR, the DL-3400 and the DL-3282. Finally, using the acquired knowledge from the present work, one can look back at Appendix A in [1] and identify other radios which could be potentially reverse-engineered, which is worrisome. Most of them are using FSK modulation but integrate additional signal processing techniques, such as Hamming Code FEC and Cyclic Redundancy Checks (CRCs). The capability to reverse-engineer such radios would be very interesting to investigate in the future. Our work demonstrates that the underlying assumption when those SCADA radios were designed—that usage of an obscure and secret modulation format was secure enough—no longer holds given the widespread availability of low-cost SDR hardware and associated knowledge.

References

1. Roark, R.C., Van Wie, D.G.: Feasibility Study of a New Air Interface and Physical Layer Packet Definition for the ALERT User Community (2003)
2. Boyes, W.: Instrumentation Reference Book, 4th edn, p. 27. Butterworth-Heinemann (2011). ISBN 978-0-7506-8308-1
3. Siggins, M.: 14 Major SCADA Attacks and What You Can Learn From Them. DPS Telecom. Accessed 26 Apr 2021
4. Reverse-Engineering Wireless SCADA Systems. https://shmoo.gitbook.io/2016-shmoocon-proceedings/build_it/10_reverse-engineering-wireless-scada-systems. Accessed 14 Dec 2020
5. FCC Id NP42424016-001 - CalAmp Wireless Networks Corporation T-96SR Telemetry Transceiver/Modem. https://fccid.io/NP42424016-001. Accessed 16 July 2022
6. Blossom, E.: GNU radio: tools for exploring the radio frequency spectrum. Linux J. (2004)
7. Attacks targeting industrial control systems is up 110 percent. https://securityintelligence.com/attacks-targeting-industrial-control-systems-ics-up-110-percent/. Accessed 29 Aug 2022
8. Global Cyber Megatrends. https://www.raytheon.com/sites/default/files/2018-02/2018_Global_Cyber_Megatrends.pdf. Accessed 29 Aug 2022
9. ADALM-PLUTO Software-Defined Radio Active Learning Module. https://www.analog.com/en/design-center/evaluation-hardware-and-software/evaluation-boards-kits/adalm-pluto.html. Accessed 23 Nov 2022