How reversing the COMBUS protocol resulted in breaking security of a security system

16.11.2018.
IT-SECX 2018, Austria

Author

- Lead researcher at Possible Security, Latvia
- Hacking and breaking things
  - Network flow analysis
  - Reverse engineering
  - Social engineering
  - Legal dimension
- twitter / @KirilsSolovjovs

INTRO

Paradox security systems

- Canadian company, founded 1989
- Modular security alarms
  - SPECTRA SP
    - Expandable Security Systems
  - EVO
    - High-Security & Access Systems
  - MAGELLAN
    - Wireless Security Systems

Prior research

- Work on interfacing with SP series via COMBUS
  - Martin Harizanov
    - partially working code, moved on to SERIAL
- Work on interfacing with MG series via SERIAL
  - All over forums
    - leaked docs
  - Gytis Ramanauskas
    - code on github

Responsible disclosure process

- At first:
  - General claim that there's a vulnerability met with doubt
  - Clearly no process in place
- In a few of months:
  - The information has been "dealt with"
  - For obvious security reasons, it is our policy to never discuss engineering matters outside of the company and thus we will not be commenting further on this issue
- Now doing public disclosure a couple years later

¯_(ツ)_/¯

Components

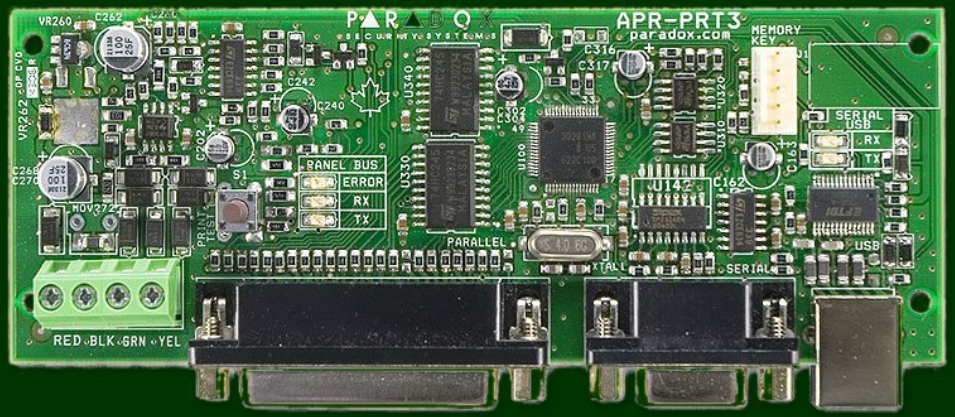
- **zone** interrupt devices
- **PGM** modules
- **serial** devices
- **ancillaries**

Components

- **combus slaves**

  provide two-way communication

  - keypads
  - modules
    - expansion
    - printer
    - listen-in
    - etc.

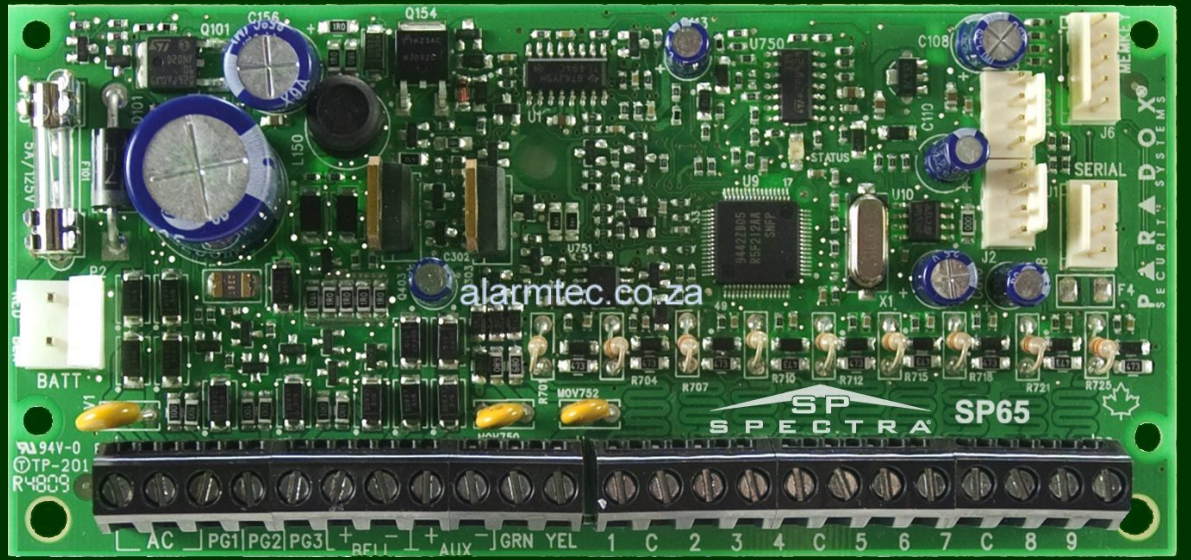


Components

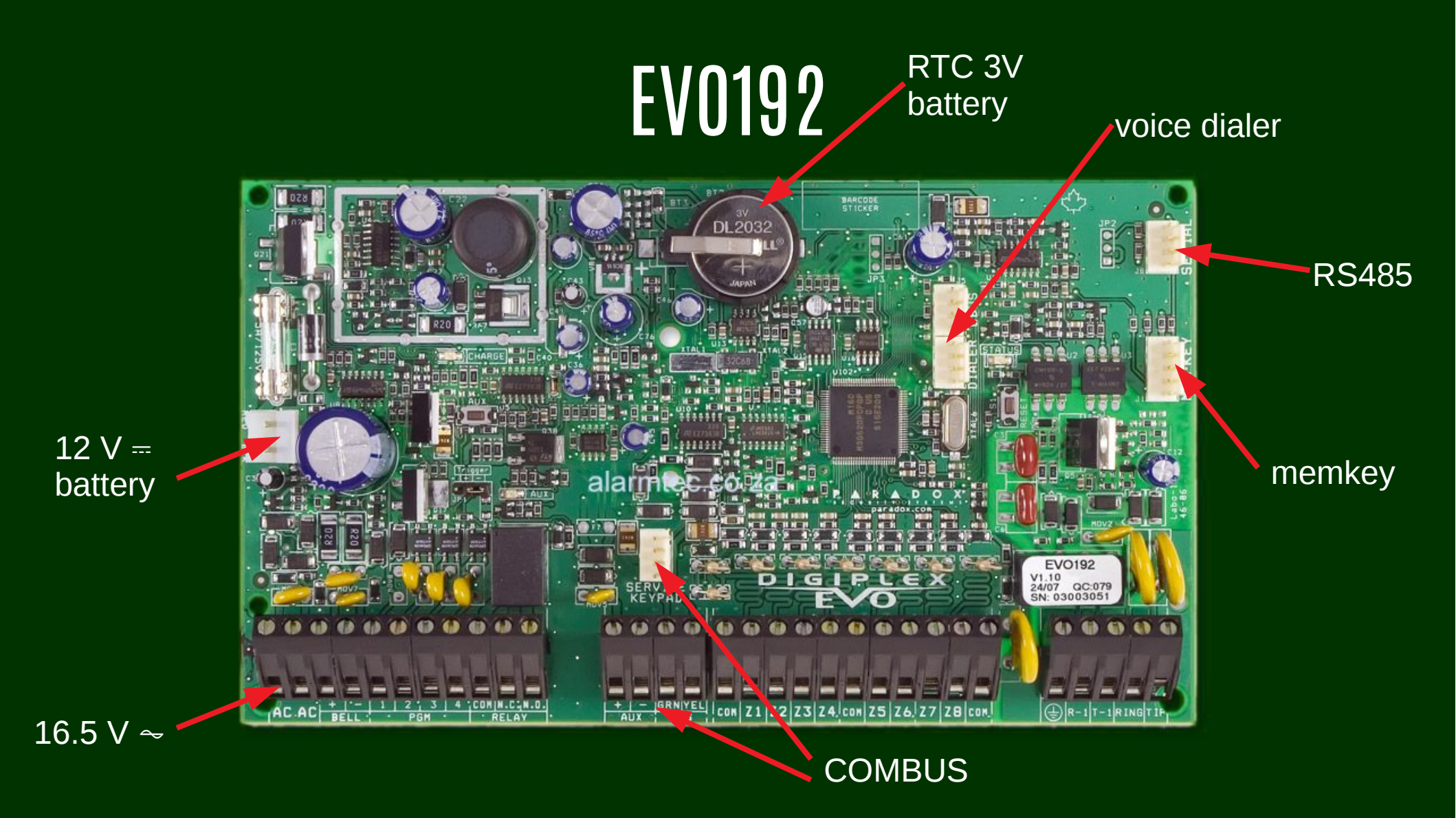
- **master**

  heart on the system – “motherboard”

  - panel



EVO192


RTC 3V battery
voice dialer
RS485
12 V ⎓ battery
memkey
16.5 V ≂
COMBUS
alarmtec.co.za
DIGIPLEX EVO
EVO192
V1.10
24/07 QC:079
SN: 03003051
AC AC
BELL
PGM
RELAY
AUX
Z1 Z2 Z3 Z4 Z5 Z6 Z7 Z8
R-1 T-1 RING TIP

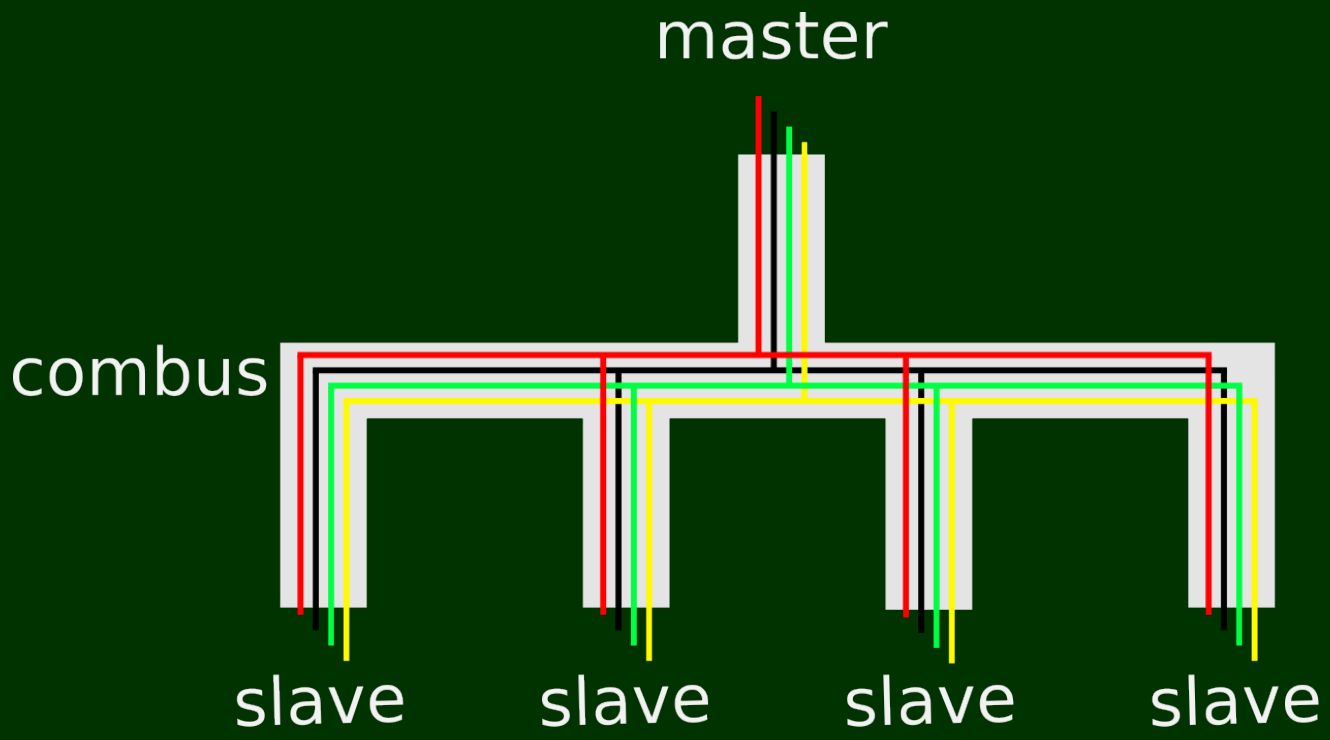
REVERSE ENGINEERING

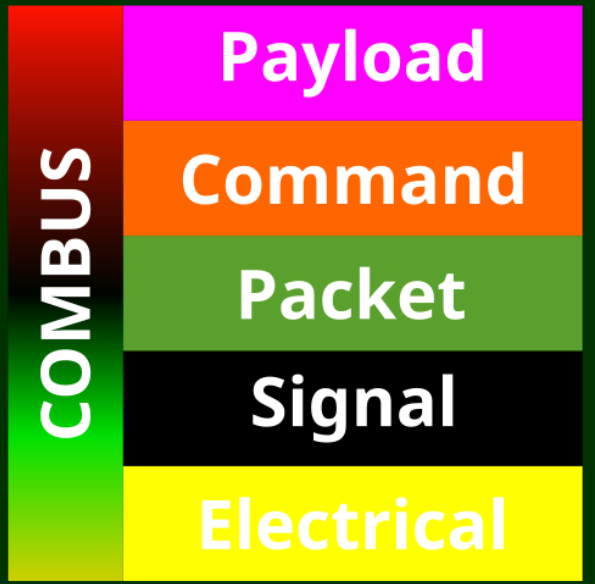
Hardware tools

- Saleae Logic 8

- Arduino UNO

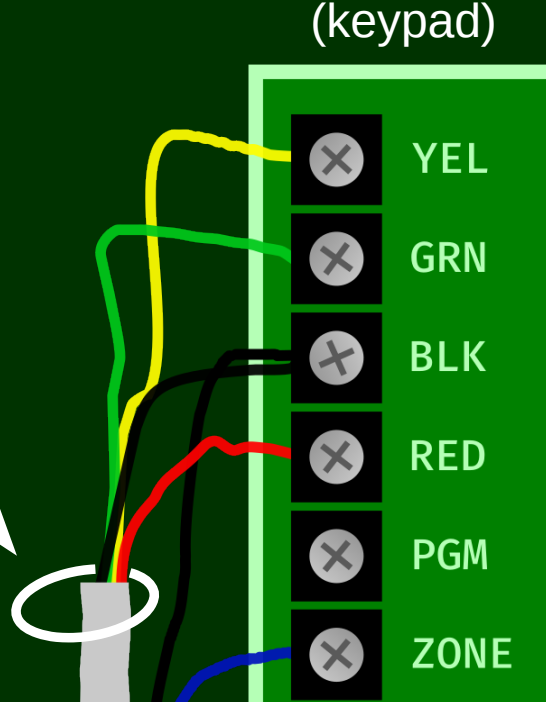
COMBUS


master
combus
slave
slave
slave
slave


COMBUS
Payload
Command
Packet
Signal
Electrical

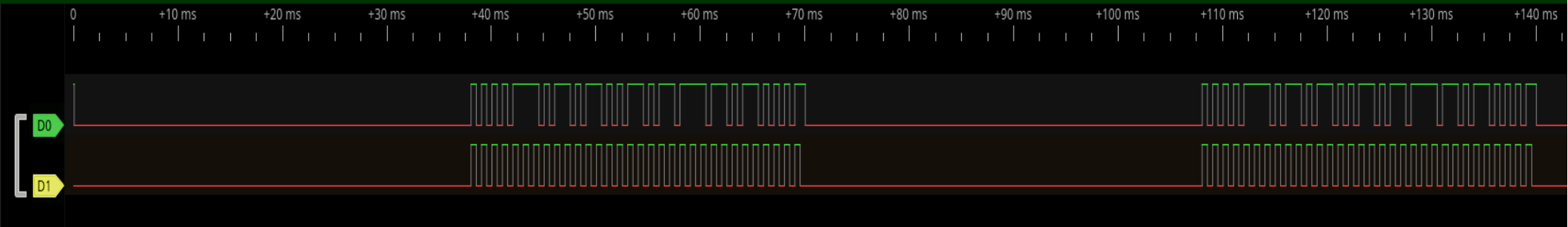
Electrical layer

- combus – 4 wire bus
- resistance = 0 ⇒ black = GROUND
- stable⎓voltage ⇒ red = POWER
- … ?



Signal layer

- yellow = CLOCK
- green = DATA
- 40ms between packet bursts
- 1 clock cycle = 1ms; signal = 1kHz

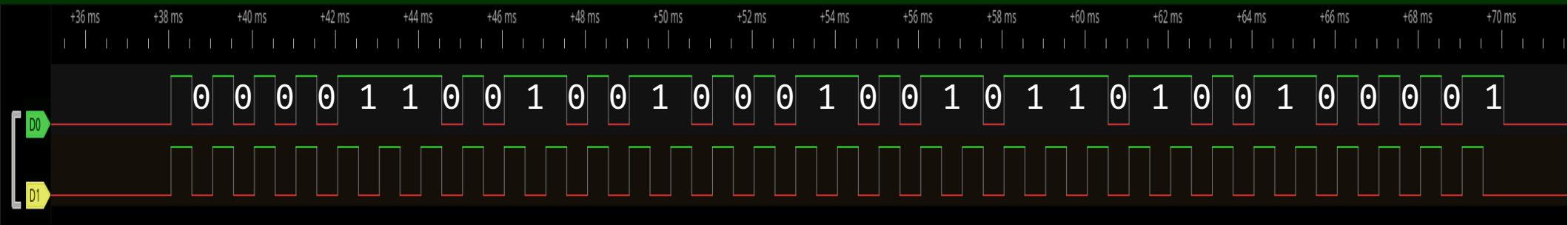


Signal encoding

- CLOCK = low ⇒ data!!! ☺
- … we should have two-way comms

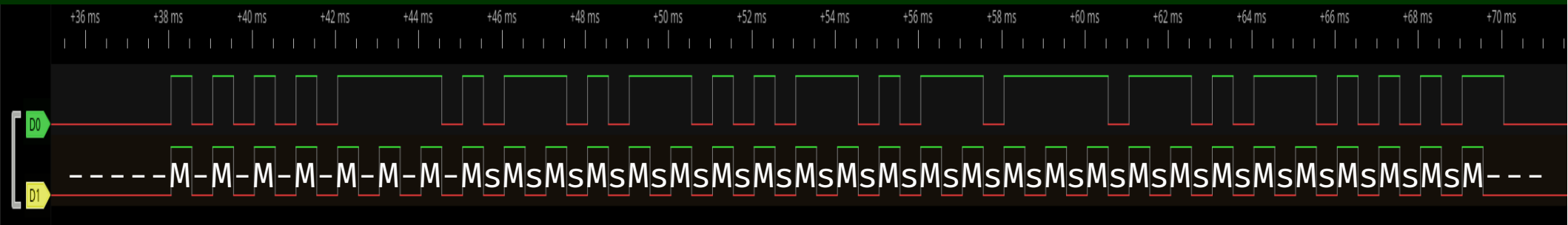
  something is missing ☹

0 C 9 1 2 D 2 1



Full signal encoding

- CLOCK = high
  - slave pulls down to send “1”
- CLOCK = low
  - master pulls up to send “1”



Packet structure

01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20	21	22	23
master																						
40	03	92	02	01	EB	01	00	00	00	00	00	00	00	00	00	00	00	00	00	00	C4	~~00~~
E2	14	10	0B	0F	37	05	00	01	5D	~~00~~												
0C	13	38	1B																			
slave																						
~~00~~	02	20	00	00	00	FF	5A	22	00	00	00	00	D5	23	79	E2	00	00	00	C8	B6	00
~~00~~	02	00	00																			

command checksum ~~unused~~ channel-request

checksum – SUM mod 0x100, starts at command

Commands: heartbeat / clock

- 0C AA 10 11
- 0C NN DD/MM HH/SS
  - NN = xxxxxxxp = sequence number
- p==0 → 0C NN DD HH
  - DD = day of the month
  - HH = hour
- p==1 → 0C NN MM SS
  - MM = minutes
  - SS = seconds

Commands: code entry

- 00 02 20 00 00 00 FF **12 34** 00 00 00 00 D9 10 3A 99 00 00 00 00 21 00
- 00 02 20 UT 00 00 CT CC CC 00 00 00 00 SS SS SS SS 00 00 00 00 ## 00
  - UT = pxxxxxxx
    - p = user type == 1 → programmer
  - CT = code type
  - CC CC = code (oh, check this out, it looks like a code)
  - SS SS SS SS = serial number of source device
  - ## = checksum

Payloads

- No encryption used
- Text as fixed length (often 16 chars) ASCII strings
  - 0x20 = filler

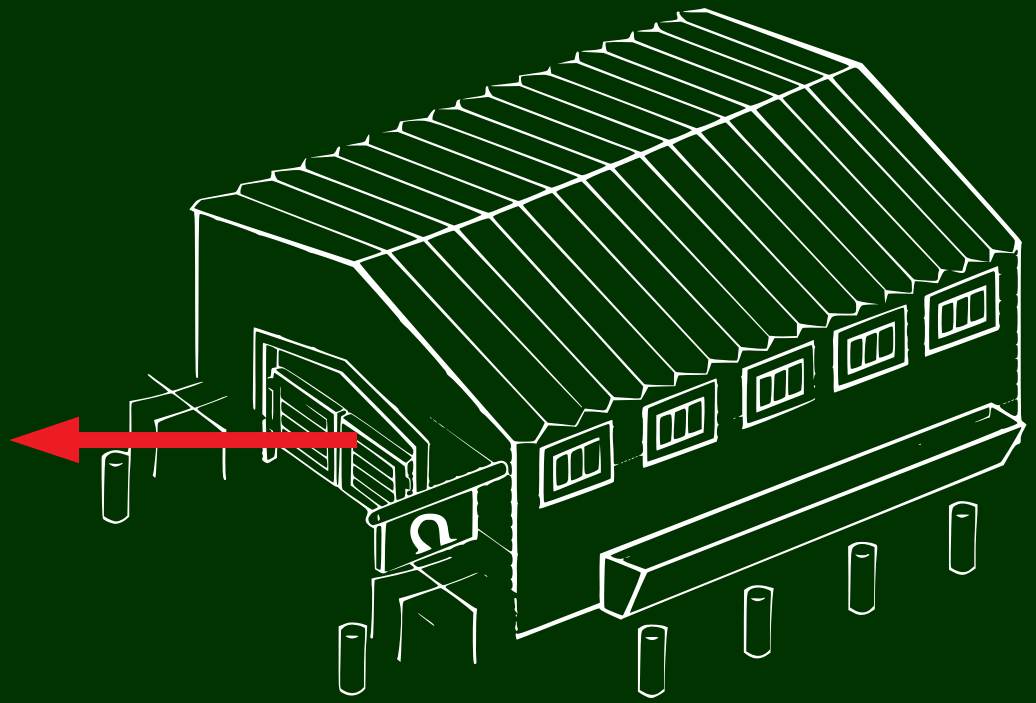
```
b0 02 00 00 00 44 6f 6f | .....Doo
72 20 30 31 20 20 20 20 | r 01
20 20 20 20 20 e7 00    |       ..
```

- Numbers usually packed BCD
  - “0” is 0b1010 = 0xA
  - no encryption, but hey, at least we got obfuscation!

DEMO TIME

Before connecting a module to the combus, remove AC and battery power from the control panel.

Exploitation scenarios



```
3998 3111 9309 1400
8248 4584 9450 5617
6550 8245 6979 9878
6101 4971 1294 9576
5005 2789 7113 3627
6856 5132 4920 5076
7500 7065 0643 9302
1744 3725 8432 1275
1128 1497 8657 9264
```

SUMMARY

Results

- Hardware built, decoding software written
- Protocol partially transcribed

Solutions

- Encryption at command layer
  - TLS
  - CA in trust-store in all components
- Mutual slave-master authentication
  - client certificates
- Sensitive payload encryption
  - with unique per-panel key (synchronized at install time)

Further research

- Anti-collision protocol research
- DoS attacks
- Emulating a slave
- COMBUS over radio
- RF attacks
- Firmware reverse engineering

Resources

- Slides available
  - http://kirils.org/
- Tools available on 18th November
  - https://github.com/0ki/paradox

How reversing the COMBUS protocol resulted in breaking security of a security system

16.11.2018.
IT-SECX 2018, Austria

http://kirils.org/ @KirilsSolovjovs