Horror on the bus

Hacking combus in a Paradox security system

Hack In The Box Dubai 2018

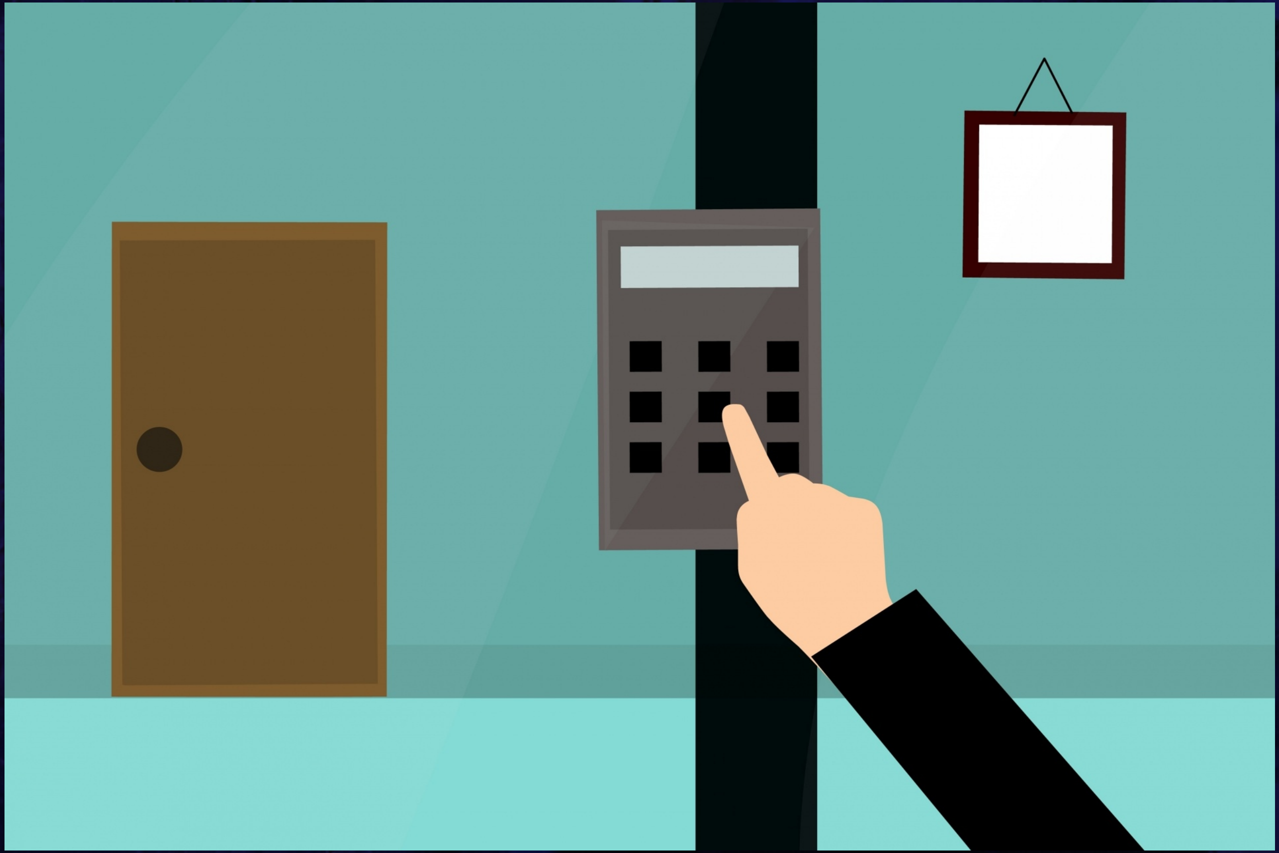
#HITB2018DXB

#HITB2018DXB

Author

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

Security alarm systems



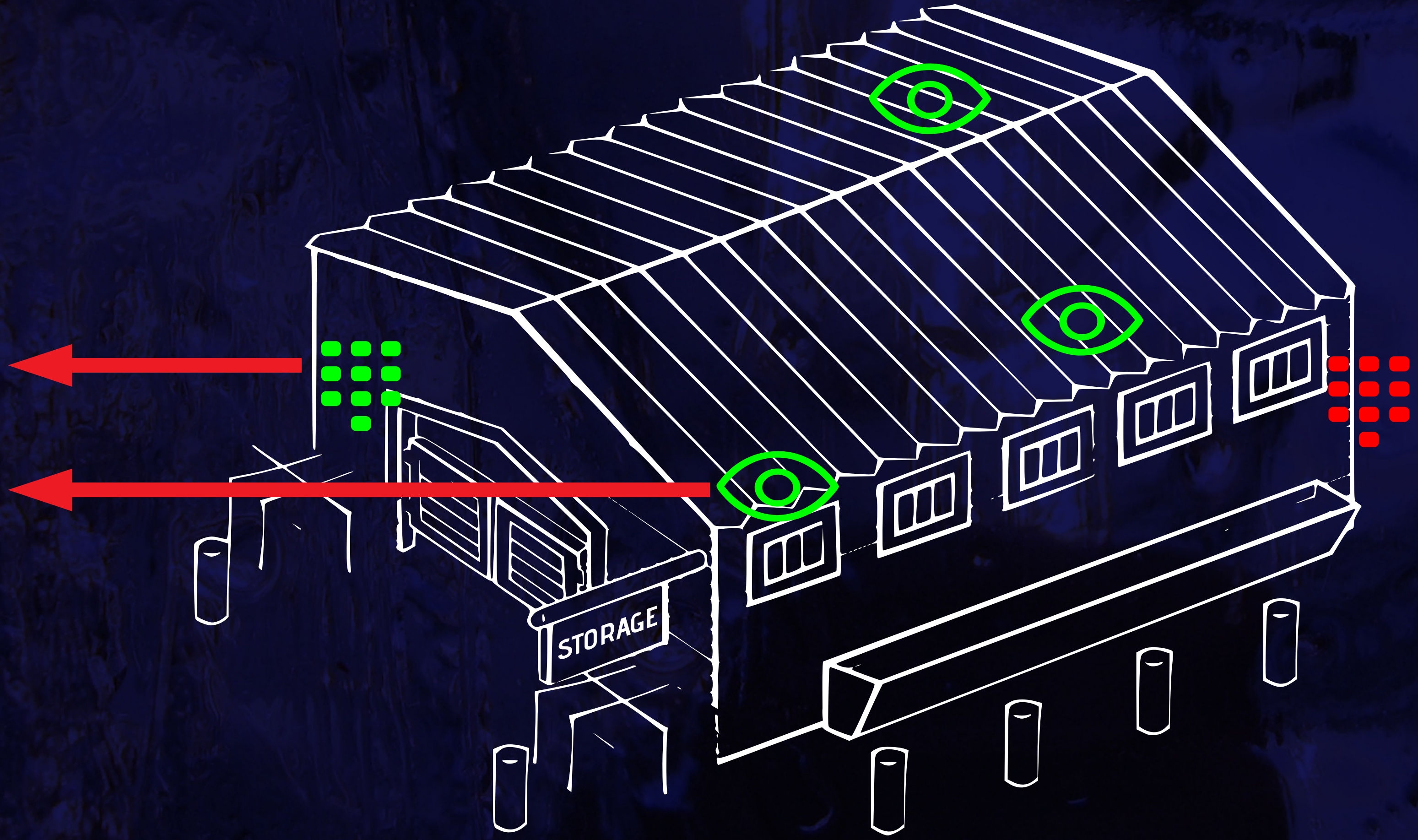
#HITB2018DXB

Security alarm systems

What could go wrong?

```
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
```

Does this provide a peace of mind?



INTRO

#HITB2018DXB

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»
- A couple years later I’m doing public disclosure

¯_(ツ)_/¯

Components

- **master**

  heart on the system – “motherboard”

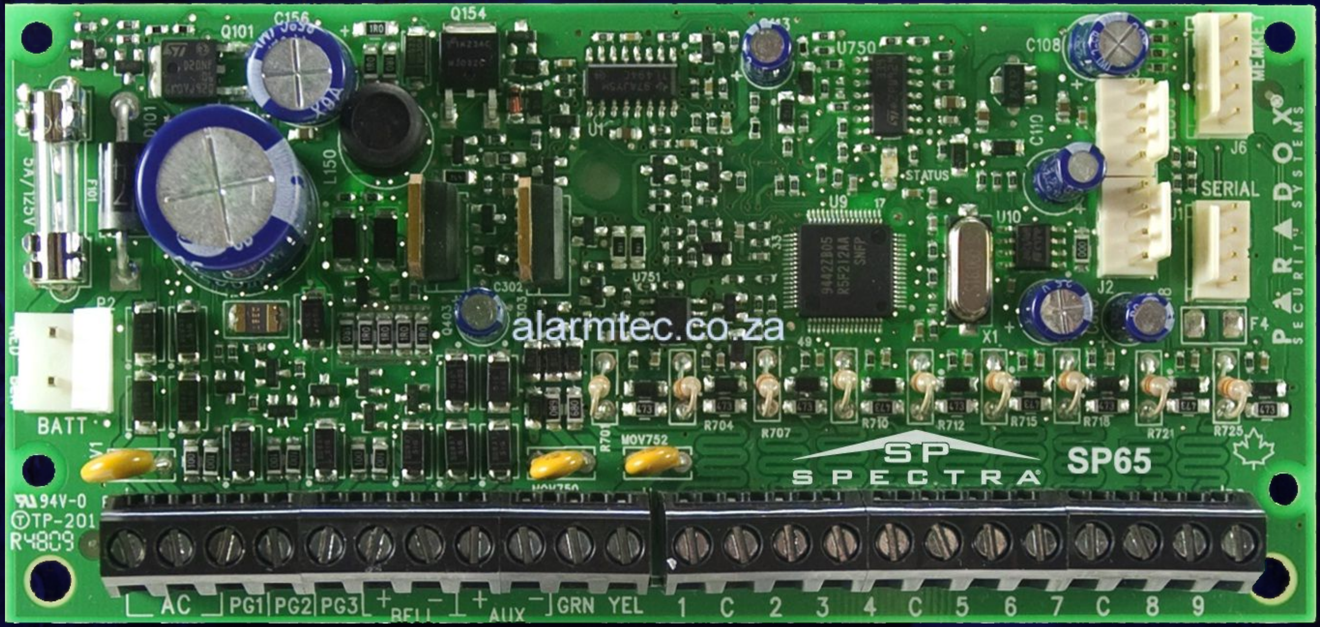
  – panel

- **ancillaries**

  – battery

  – power supply

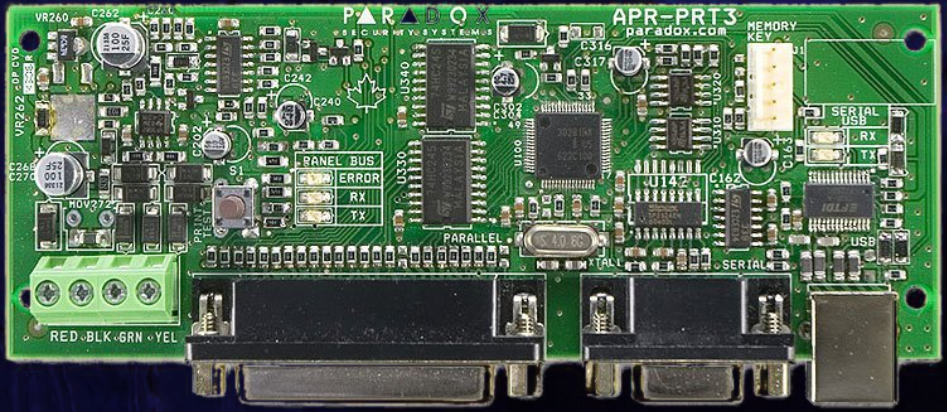
  – siren



Components

- **combus slaves**

  provide two-way communication
  - keypads
  - modules
    - expansion
    - printer
    - listen-in
    - etc.



Components

- **zone** interrupt devices

  input, measures resistance ⇒ chaining
  - magnetic sensors
  - PIR sensors
  - panic buttons
  - etc.

Components

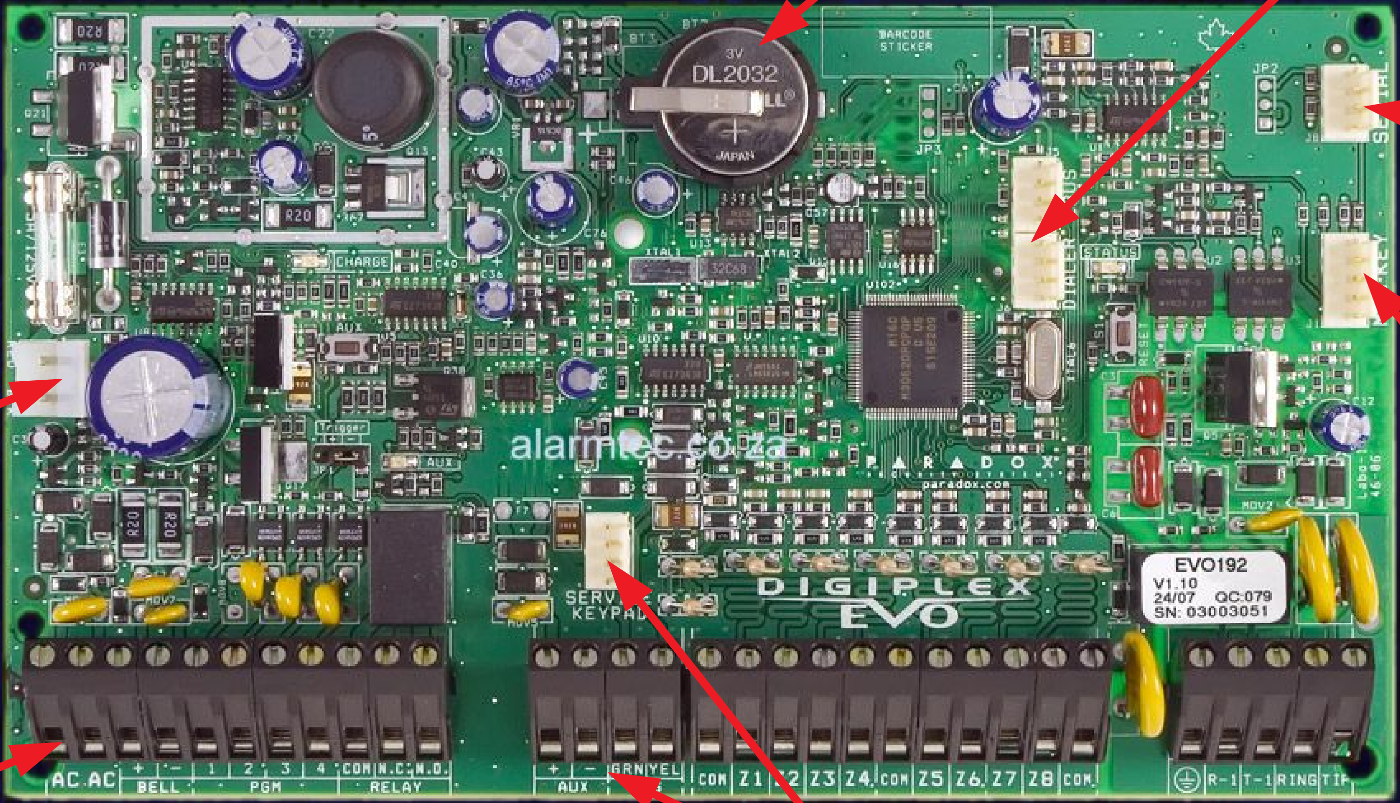
- **PGM** modules:

  output, 100mA relays (solid state)
  - external actuators
  - boost relays

Components

- **serial** devices:
  - RS485
  - Serial converters (RS232, usb)
  - IP modules
  - GSM modules
  - etc.

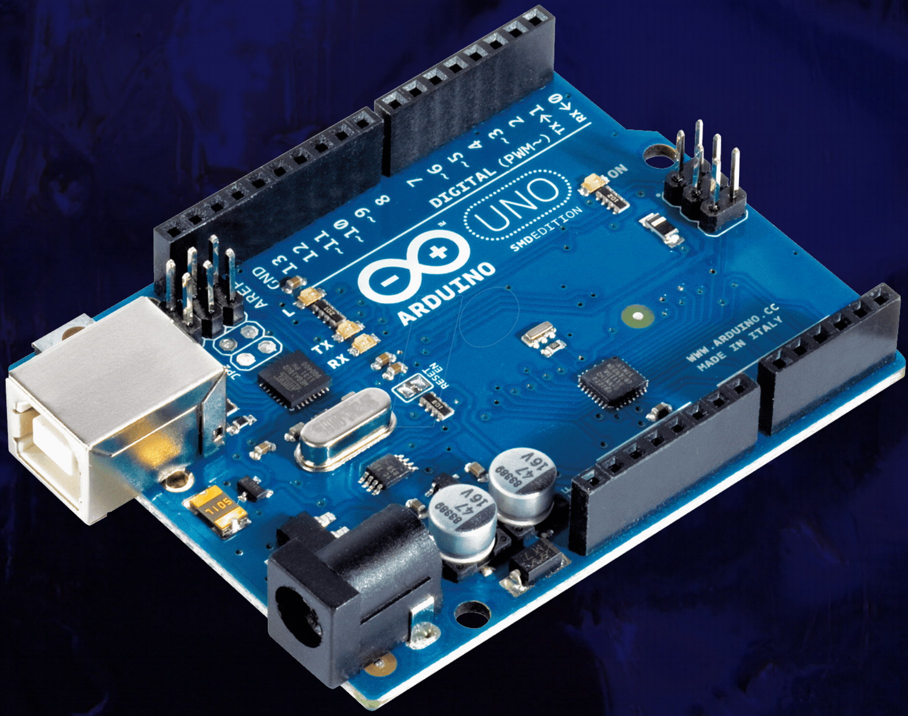
EVO192


RTC 3V battery
voice dialer
RS485
memkey
12 V ⎓ battery
16.5 V ∼
COMBUS

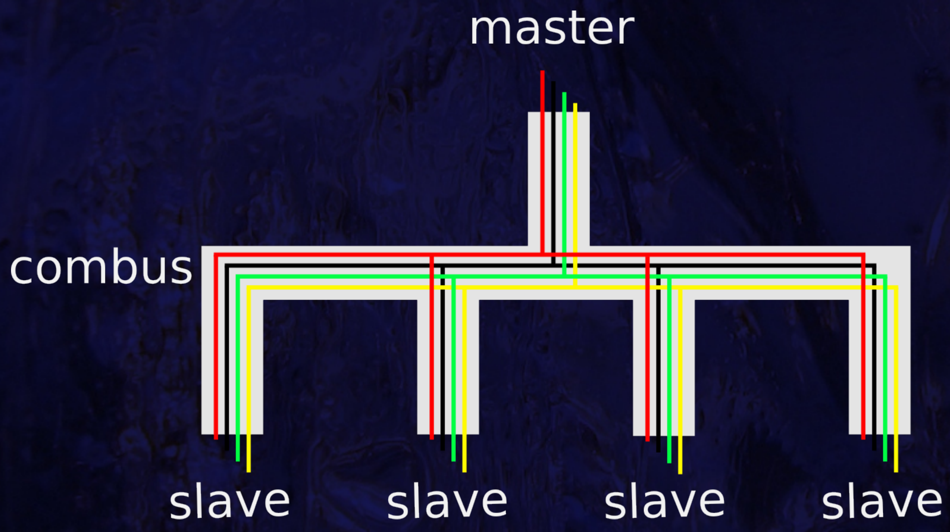
REVERSE ENGINEERING

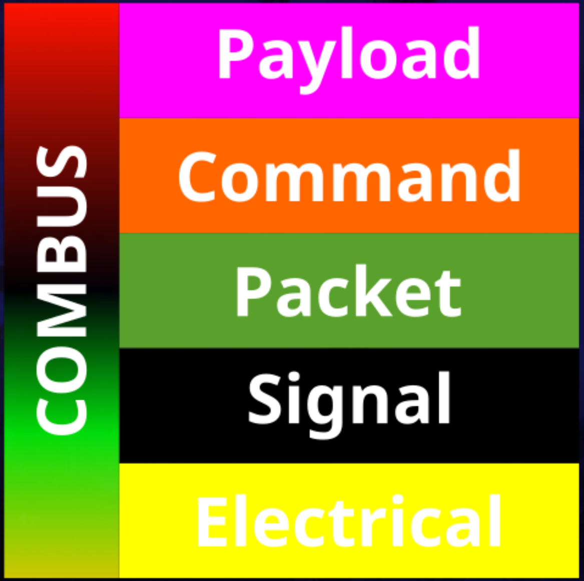
Hardware tools

- Saleae Logic 8
- Arduino UNO



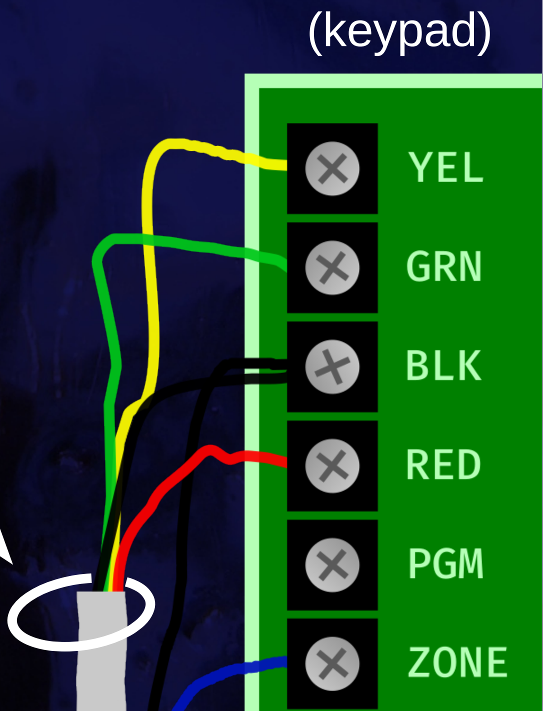
COMBUS


master
combus
slave
slave
slave
slave


COMBUS
Payload
Command
Packet
Signal
Electrical

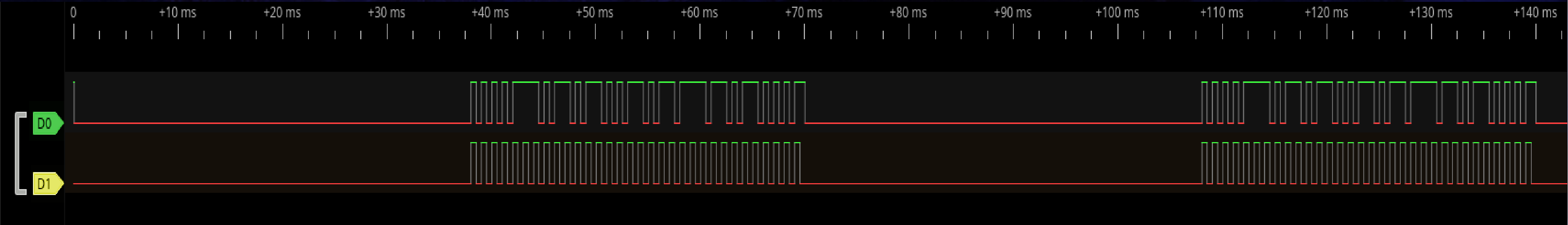
Electrical layer

- 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

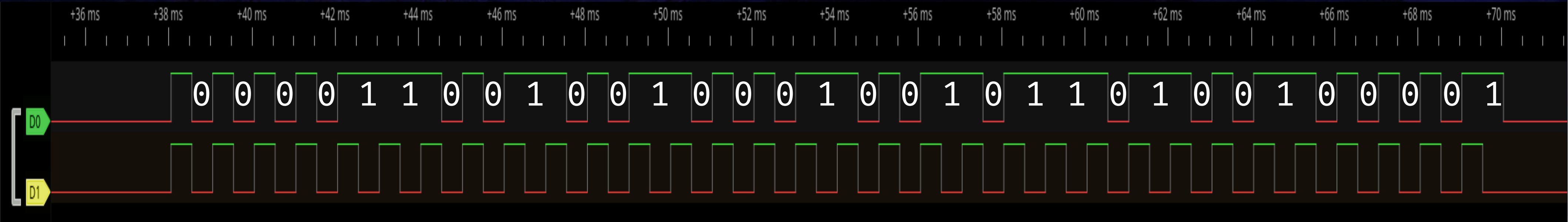


Signal encoding

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

  something is missing ☹

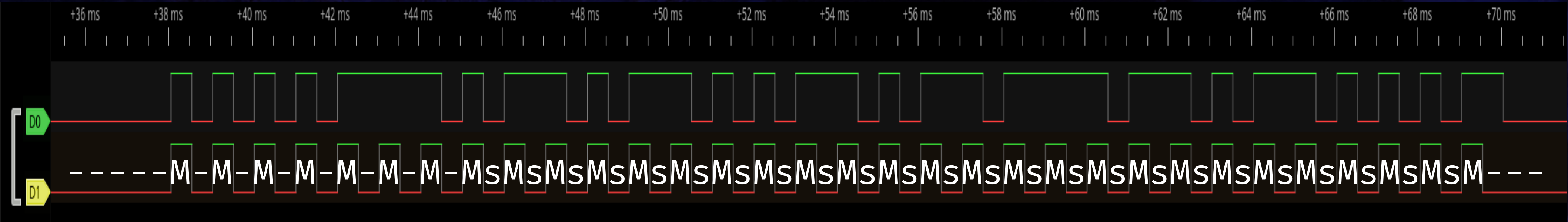
0 C 9 1 2 D 2 1



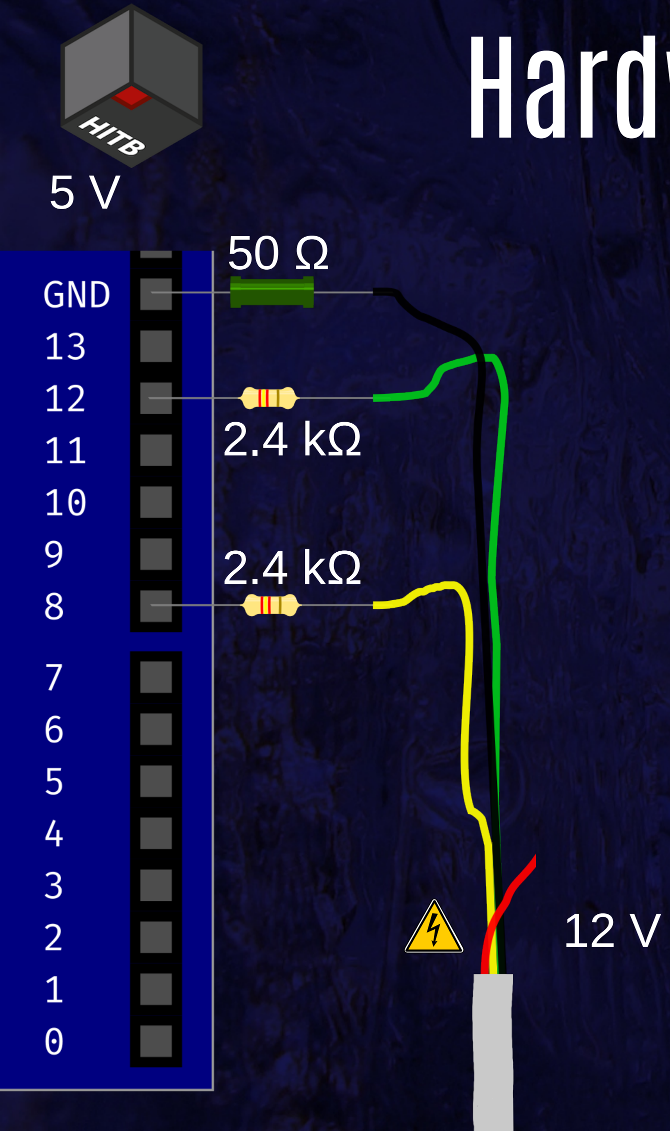
#HITB2018DXB

Full signal encoding

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

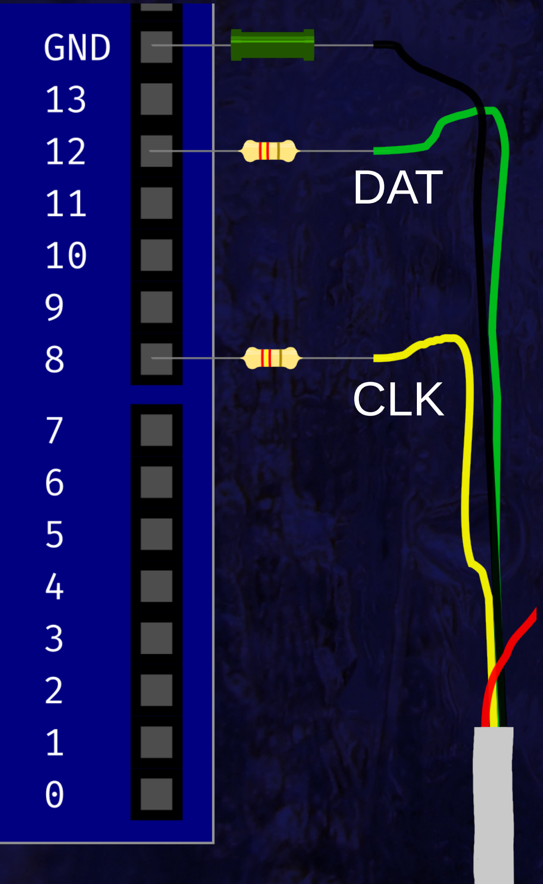


Hardware setup (read-only)



- Resistors to limit
  - voltage
  - current draw

Decoding into bytes



```
on CLK change:
  wait 50µs
  if CLK == high:
    master ← master<<1 +  DAT&1
  else:
    slave  ← slave<<1  + !DAT&1

on idle > 2ms:
  if master > 0:
    print master
    print slave
    master ← 0
    slave ← 0
```

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
E2 14 10 0B 0F 37 05 00 01 5D ~~00~~
0C 13 38 1B

slave
~~00~~ 02 00 00
~~00~~ 02 20 00 00 00 FF 5A 22 00 00 00 00 D5 23 79 E2 00 00 00 C8 B6 00
```

command checksum ~~unused~~ channel-request

Checksum

```
checksum ← 0
for i in @command to @checksum - 1:
    checksum ← (checksum + *i) % 100
```

Commands: heartbeat / clock

- 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

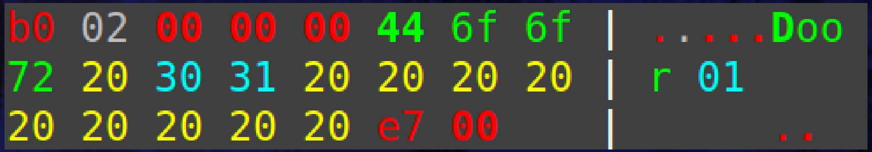
```
0C AA 10 11
```

Commands: code entry

- 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
  - SS SS SS SS = serial number of source device
  - ## = checksum

```
00 02 20 00 00 00 FF 12 34 00 00 00 00 D9 10 3A
99 00 00 00 00 21 00
```

Payloads



```
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    |       ..
```

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

EV0192

“Digiplex and Digiplex EVO systems provide the highest level of protection for banks, high-security military and government sites, luxurious residential homes and any place where maximum security is essential”

– https://www.paradox.com/Products/default.asp?CATID=7

SUMMARY

#HITB2018DXB

Results

- Hardware built, decoding software written
- Protocol partially transcribed
- Impact of possible attacks

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

- DoS attacks
- Emulating a slave
- COMBUS over radio
- RF attacks
- Firmware reverse engineering

Horror on the bus

Hacking combus in a Paradox security system

28.11.2018.
Hack In The Box Dubai 2018

http://kirils.org/
https://github.com/0ki/paradox

@KirilsSolovjovs