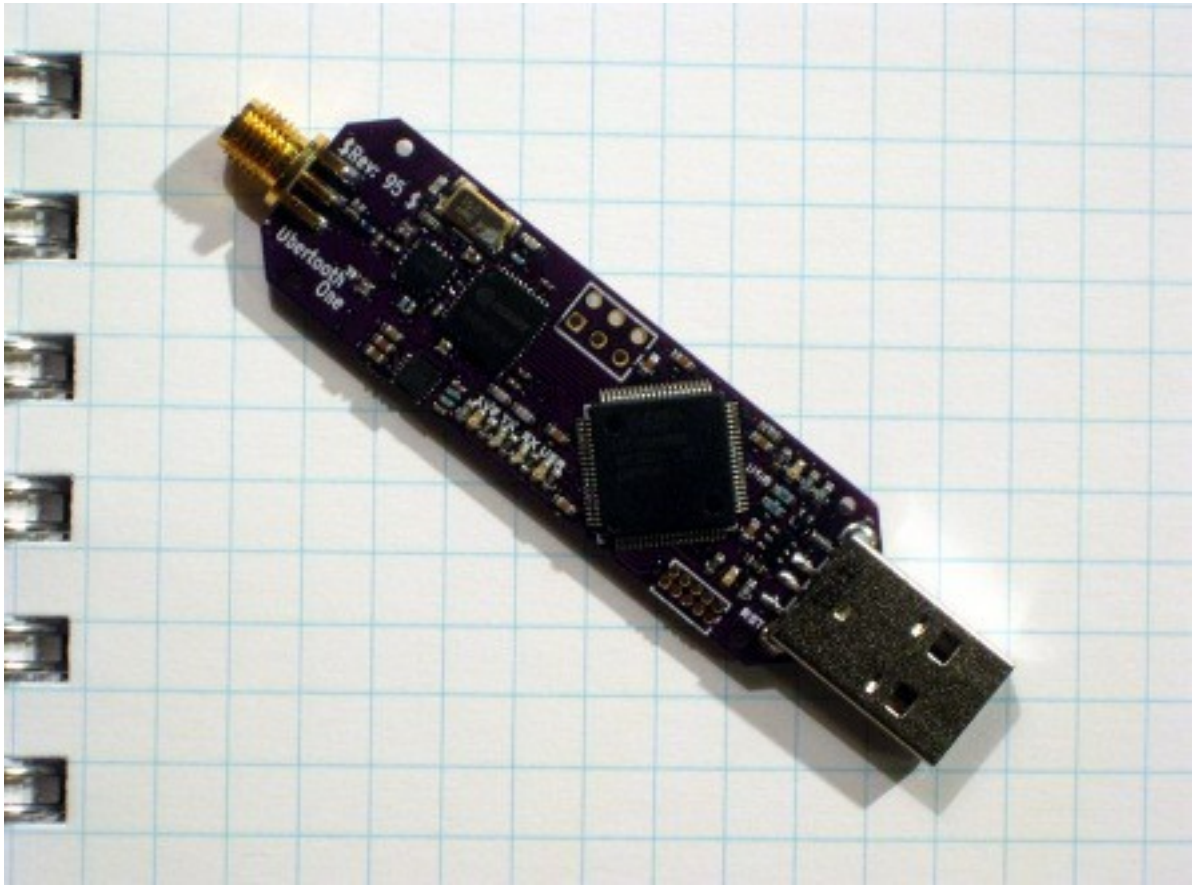
Bluetooth Packet Sniffing Using Project Ubertooth

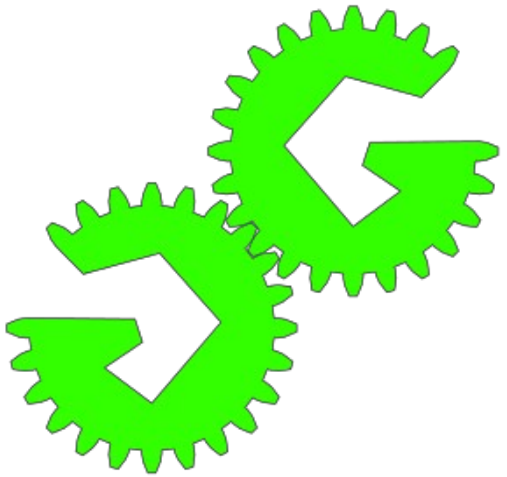


Dominic Spill
dominicgs@gmail.com

Dominic Spill

- Bluesniff: Eve meets Alice and Bluetooth
  - Usenix WOOT 07
- Building a Bluetooth monitor
  - Shmoo/Defcon/Toorcamp 09
  - With Michael Ossmann
- Lead on project Ubertooth

Disclosure

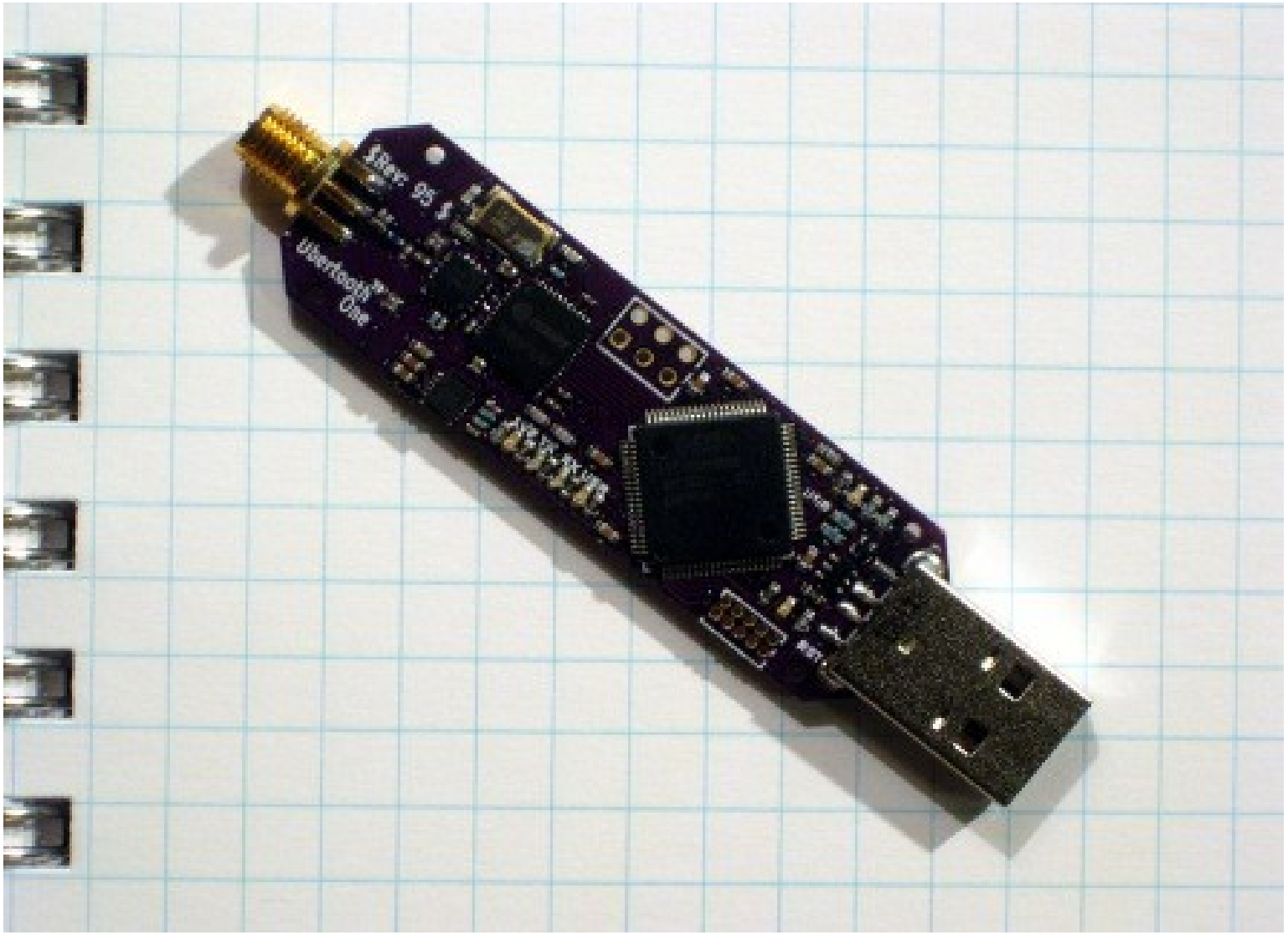


- Not an employee of GSG
- I receive some funding
- Not here to sell Ubertooths

Warning

- If you wish to remain anonymous:
  - Remove your name from Bluetooth device names
  - Or turn off Bluetooth devices now
- Live demos at a con may not work
  - Especially when using 2.4GHz

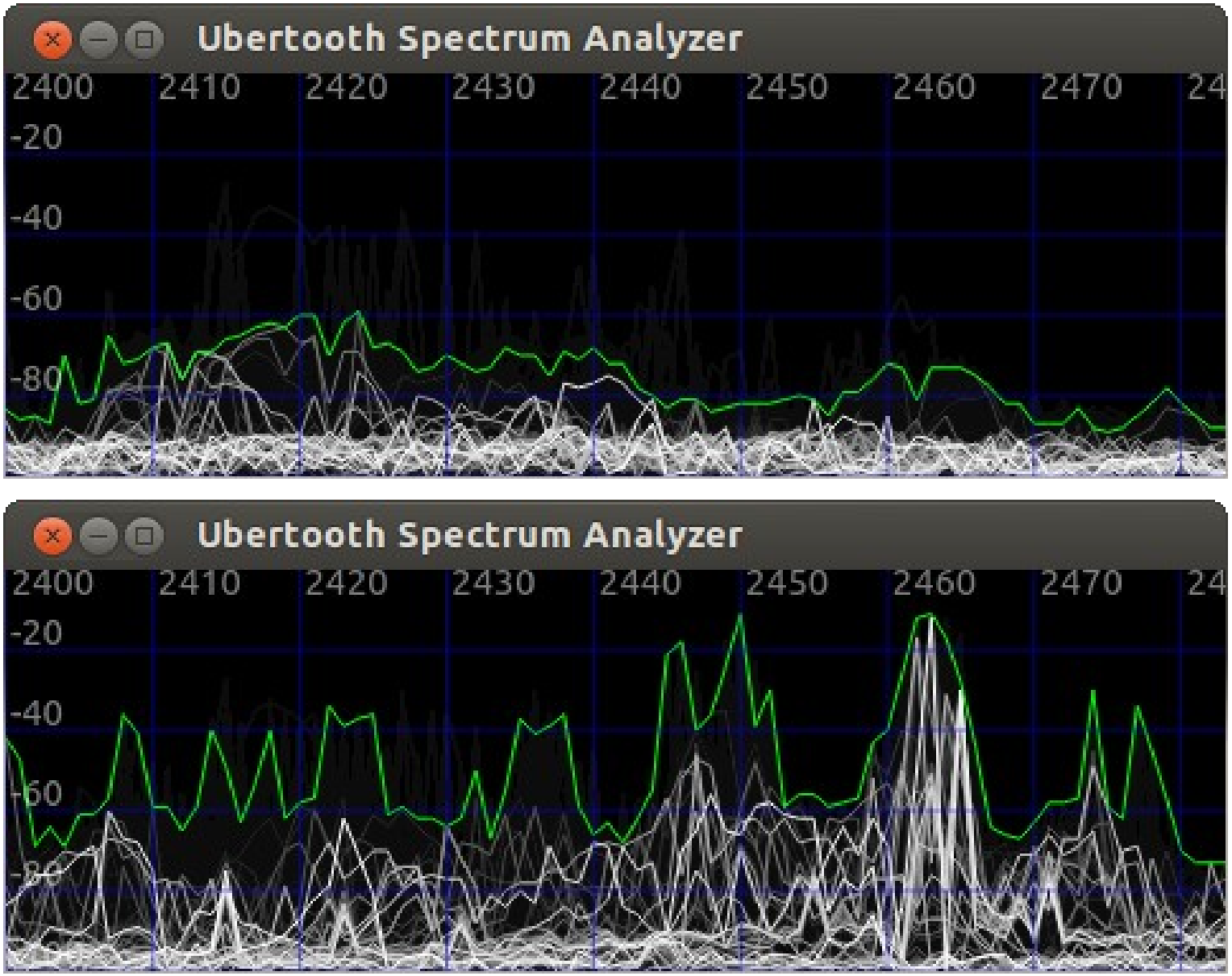
Ubertooth



Ubertooth

- Designed by Michael Ossmann
- 2.4GHz experimentation platform
- Bluetooth 1.x, Low energy, 802.11 FHSS
- Hardware
  - CC2400 (+CC2591 frontend)
  - NXP LPC1756
  - USB device (2.0)
- Open source software and hardware
  - http://ubertooth.sourceforge.net

Spot the difference?



Bluetooth

Bluetooth

- 2.4GHz ISM band
- Variable data rates
  - Basic Rate – 1Mb/s
  - Enhanced Data Rate – 3Mb/s
  - High Speed - Alternate MAC/PHY – 24Mb/s
  - LE (Smart) – 200Kb/s
- FHSS @ 1600Hz
  - 79 channels

Bluetooth

- Bluetooth SIG
  - 17,000 members
  - Free to join
- Bluetooth devices
  - 7 billion devices sold to end 2011
  - Will ship 2 billion devices this year
  - 20 billion expected in use by 2017

http://www.bluetooth.com/Pages/sig-membership.aspx

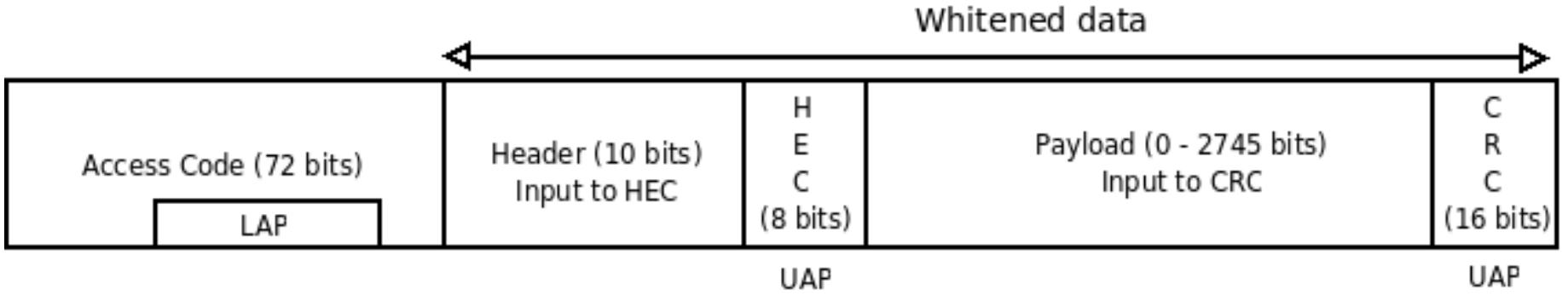
Bluetooth - Terminology

- Bluetooth device address / MAC / BD_ADDR
  - Three parts, not all present in packets
    - LAP - Lower – lowest 24 bits
    - UAP - Upper – next 8 bits
    - NAP - Non-significant – top 16 bits

- CLKN
  - 27bit 3200Hz internal clock
  - Increments twice per time slot

Bluetooth - Terminology

- Access code
  - Derived from LAP
- Packet Header
  - Error check based on UAP
- Payload
  - Possibly encrypted
  - CRC also based on UAP



Bluetooth - Terminology

- Non-Discoverable mode
  - Does not respond to inquiry scans
  - Still responds to page scans
  - Some newer devices ignore unknown page scans
- Data whitening
  - Packets XOR'd with pseudo-random sequence

Bluetooth sniffing is hard

- No “monitor mode”
  - Fixed correlator – not promiscuous
- Frequency hopping
  - 1600 hops/s
  - 625us/packet
  - Pattern based on MAC and CLKN
- Data whitening
  - PRNG initialised with CLK1-6

Bluetooth sniffing is profitable (apparently)

- Known connection LE only - $250
- Known connection BR only - $10,000
- All channel BR/EDR/LE - $25,000

Finding Packets – Old method

- Find access code
  - Treat 64bit chunks as possible access codes
  - LAP stored in bits 34-57
- Check access code
  - Check trailer (2 errors)
  - Generate access code from LAP
  - Compare access code to 64bit chunk (6 errors)

Packets!

00001011111010111010101010101010101010101010101
01010101010101010110111111001100001011011001100110
01111111001101001011000111100101010111111111000000111
10000000000000000000111111111000000111110000111100
00111100001111000011110000111100001111000011110000111
11100001111000011110000111100001111000011110000111100
00001111000011110000111100001111000011110000111100000
01111000011110000111100001111000011110000111100000000
00100001111100010101001110001110111011100111011010011
10100010000100010101010

Flaws

- Slow on desktop CPU
- Unworkable on low power devices
- No errors allowed in LAP
- No error correction

Error Correction

Error Correction

- (64, 30) expurgated block code
  - Based on BCH (63, 30) code
  - Calculate syndromes to find error vectors
- Supposed to correct up to 6 bit errors
  - Too many false positive results
  - In practice correct <4 bit errors

Error Correction

- Manufacturers don't implement it
  - Known access code loaded into correlator
  - Compared to received bits
  - Up to 6 bit errors
- This is what we do for a known address

Finding Packets – New Method

- Pre-calculate syndromes for n-bit errors
  - Use known access code
  - XOR with all possible n-bit error vectors
  - Generate syndrome for each error
  - Store in hash (uthash rules!)
- For each 64bit block
  - Calculate syndrome
  - Check hash for error vector
  - Correct error

Finding Packets – New Method

Demo

Ubertooth-scan

- Finding non-discoverable devices

- Wright's Law
  - Security will not get better until tools for practical exploration of the attack surface are made available.

Frequency Hopping

Frequency Hopping – Local Device

- Ubertooth-follow
  - Follow a local Bluetooth device
  - Use bluez to extract CLKN
  - Push to Ubertooth
  - Start hopping

- Demo

Frequency Hopping – Local Device

- Pros
  - Reliable
  - Potentially sniff pairing
- Cons
  - Requires local BT device
  - No AFH support
    - Expected soon
  - Clock drift causes problems
    - This is fixable

Frequency Hopping – Any Device

- Derive CLKN from received packets
  - Calculate hopping pattern for known address
  - Sniff single channel or hop randomly
  - Observe packets, timing and channel
  - Place packets in hopping pattern
  - Yields unique CLKN
- Calculate clock offset from CLKN → Ubertooth
- Send to Ubertooth
- Follow hopping piconet

Frequency Hopping – Any Device

- Ubertooth-hop
  - Follow a remote piconet
  - Given LAP and UAP
  - Finds clock offset and hops
- Demo

Kismet Plugin

- Plugin for current and upcoming Kismet
  - Only survey mode – static or sweep
- Demo

Wireshark Plugin

Demo

Bluetooth Smart

- AKA
  - Bluetooth Low Energy
  - Bluetooth 4.0
  - Wibree
- Much simpler protocol
- Mike Ryan has just started working on this
  - Sniffing connection phase
  - Sniffing some data
  - AES Encryption – possible flaws in key exchange

Future Work

- Adaptive Frequency Hopping
- Encryption / Pairing
- Transmit – packet injection
- Full LE stack
- Follow in Kismet
- Storage
- Embedded platforms

Thanks to...

- Michael Ossmann
- Jared Boone
- Mike Kershaw (dragorn)
- “Will Code”
- Mike Ryan
- Zero Chaos

Questions?

dominicgs@gmail.com
Twitter: dominicgs
Slides: dominicspill.com/ruxcon/slides.pdf