Help, I’ve got ANTs!!!

whoami

- Tamas Szakaly (sghctoma)
- from Hungary, the land of Pipacs and gulash
- OSCP, OSCE
- pentester/developer @ 

- team Prauditors, European champion of Global Cyberlympics 2012
- regular speaker at Hacktivity
- gave a talk about insecure script engines in games @ DEF CON 23

what are we doing with ANTs?!

- last year I targeted one of my hobbies, flight simulators
- it was time to target the other: mountain biking
- btw, all my hobbies involve crashes…

what are we doing with ANTs?!

- lot's of gadgets involved in sports
- old-school speedometers replaced by computers
- computers that communicate with:
  - sensors (speed, power, heartrate)
  - mobile phones
  - PCs

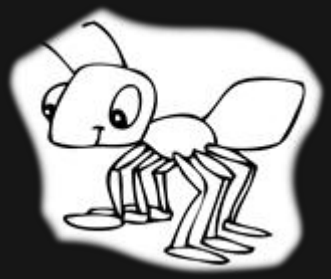
what are we doing with ANTs?!

- one of the protocols these computers talk is ANT
- also used for other purposes:
  - weight scales
  - blood pressure monitors
  - heart rate monitors
  - lots of mobile phones has ANT chips!!
  - mesh networks – FireChat (http://opengarden.com/firechat)

what are we doing with ANTs?!

- I'll talk about:
  - intro to ANT, ANT+ and ANT-FS
  - protocol-level design errors
  - implementation errors

plain old ANT



- wireless network protocol by Dynastream
- 2.4 GHz ISM band
- designed for low-power devices
- participants in a network: nodes (slave and master)
- nodes communicate via mutually established channels

plain old ANT (channel properties)

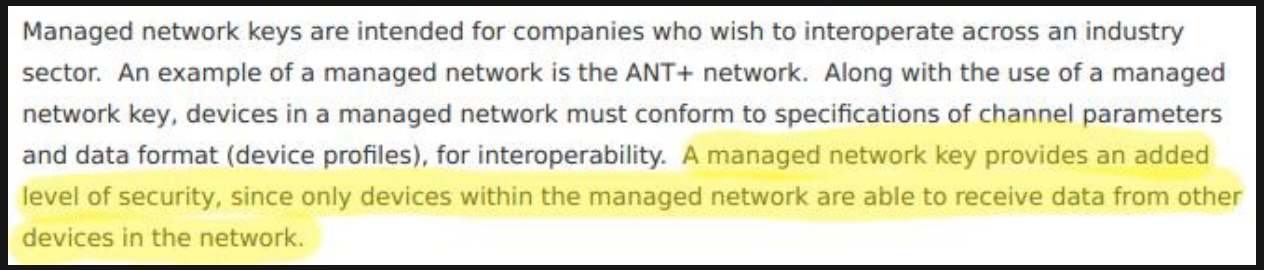
- channel type
  - bidirectional slave/master
  - shared bidirectional slave/master
  - slave receive/master transmit only
  - extended assignment
    - frequency agility
    - background scanning
    - fast channel initiation
    - asynchronous transmission

plain old ANT (channel properties)

- RF frequency (8 bit field, 2400 MHz - 2524 MHz)
- channel ID
  - transmission type (8 bit field)
  - device type (8 bit field, includes pairing bit)
  - device number (16 bit field, unique for a given type)
- channel period (16 bit field, 0.5 Hz – 200 Hz)
- network
  - network number (8 bit field)
  - network key (8 byte number)

plain old ANT (security measures)

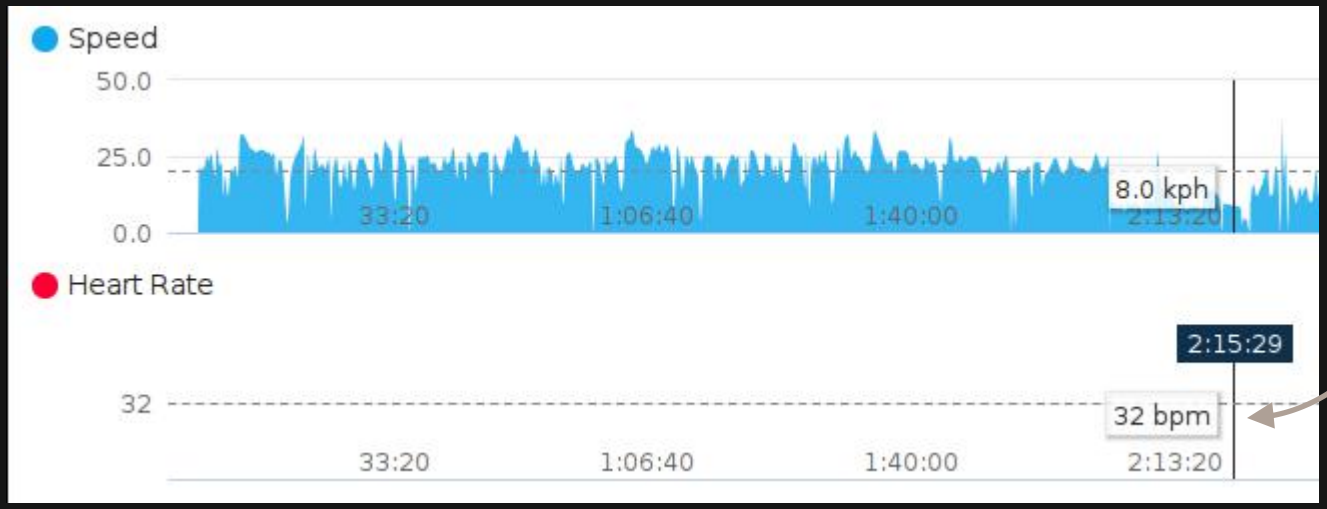
- network key


Managed network keys are intended for companies who wish to interoperate across an industry sector. An example of a managed network is the ANT+ network. Along with the use of a managed network key, devices in a managed network must conform to specifications of channel parameters and data format (device profiles), for interoperability. A managed network key provides an added level of security, since only devices within the managed network are able to receive data from other devices in the network.

- sounds good, but:
  - there is a default key (how many implementations change it?)
  - majority of gadgets use ANT+ or ANT-FS

plain old ANT (security measures)

- pairing bit
- it has to be the same on master and slave
  - bypass: open two channels with different pairing bits

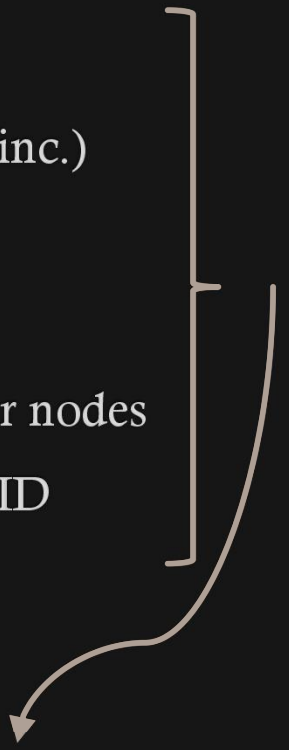


fake sensor data

- ANT test software: SimulANT+
- ANT USB dongle
- Garmin Forerunner 310XT
- Send fake power data without pairing

plain old ANT (security measures)

- inclusion/exclusion lists
  - up to 4 channel IDs on a slave node (exc. or inc.)
  - we can spoof channel ID
- white/blacklist
  - similar to inc./exc. lists, but applies to master nodes
  - so-called encryption IDs instead of channel ID



only available on recent ANT chips!!

plain old ANT (security measures)

- 128 bit AES-CTR on the channel
- key exchange???
- can't use with shared channels
- requires advanced burst data (energy-inefficient)
- can't use with ANT+

It is true that ANT protocol can use a single encrypted channel - however this is not the case for ANT+. (See the differences between ANT/ANT+ here: http://www.thisisant.com/developer/ant-plus/ant-antplus-defined)

If you use encryption for your device it is no longer ANT+ compliant and therefore you are not allowed to use the ANT+ Network Key or frequency.

atom ant (ANT+)

- built on top of ANT
- specifies so-called device profiles

ANT+ DEVICE PROFILES

Each device profile is identified by its own icon(s), which can be used on the packaging of certified devices to help consumers understand which products will work together. You can find the device profile PDFs under the documents tab of the Download page.

PWR **Bicycle Power**	CTRL Controls	GEO Geocache	SPD Multi Sport Speed & Distance	SPD Stride Based Speed & Distance
CAD Bicycle Speed & Cadence	TEMP Environment	HR Heart Rate Monitor	MOx Muscle Oxygen Monitor	SYNC Sync
BP Blood Pressure	FIT Fitness Equipment Device	LEV Light Electric Vehicle	RQT Racquet	WGT Weight Scale
EXD Extended Display	SUSP Suspension	POST Dropper Seatpost	TRK Tracker	

atom ant (ANT+)

- profiles are managed by Dynastream
- govern characteristics of ANT connection
  - frequency
  - channel period
  - network key
  - data formats
  - etc.
- managed ANT network with **publicly known network key**

GARMIN

Varia™ Rearview Rad

VISIT THE MANUFACTURER'S WEBSITE »

- World's first cycling radar that warns of vehicles approaching from behind up to 153 yards (140 me
- Tail light unit brightens and flashes to notify appro traffic of a cyclist ahead
- Tail light is controllable using ANT+bike lights cont

*This product is ANT+ Certified

ANT+ CAPABILITIES	ACTIVITIES	CATEGORIES
	Cycling	Bike Lights & Accessories

MAGURA

eLECT Dropper Seatpost

VISIT THE MANUFACTURER'S WEBSITE »

The new Vyron eLECT adds significant extra functionality and user-friendliness with its MAGURA-exclusive wireless remote operation. With a quick button press on the handlebar remote you can drop the saddle height smoothly by up to 150 mm – then another button press returns it to its optimal position for pedalling.
The Vyron uses the MAGURA eLECT Remote with ANT+ wireless technology. The same remote is already used on MAGURA suspension forks and shocks – so it's technology which has bee proven time and again over the toughest courses of the XC racing circuit.

*This product is ANT+ Certified

ANT+ CAPABILITIES	ACTIVITIES	CATEGORIES
	Cycling	Other

GARMIN

Varia™ Head Light

VISIT THE MANUFACTURER'S WEBSITE »

- Works independently and seamlessly integrates with compatible Edge® cycling computers
- As rider's speed increases, the headlight automatically projects further ahead or closer as a rider's speed decreases when paired with select Edge computers
- As light conditions change, the smart Varia headlights and tail light automatically get brighter or dimmer, when paired with a light-sensing Edge 1000
- Adjustments, including on/off, can be made manually with the use of an included remote
- With 2 tail lights you can indicate impending right or left turns

*This product is ANT+ Certified

ANT+ CAPABILITIES	ACTIVITIES	CATEGORIES
	Cycling	Bike Lights & Accessories

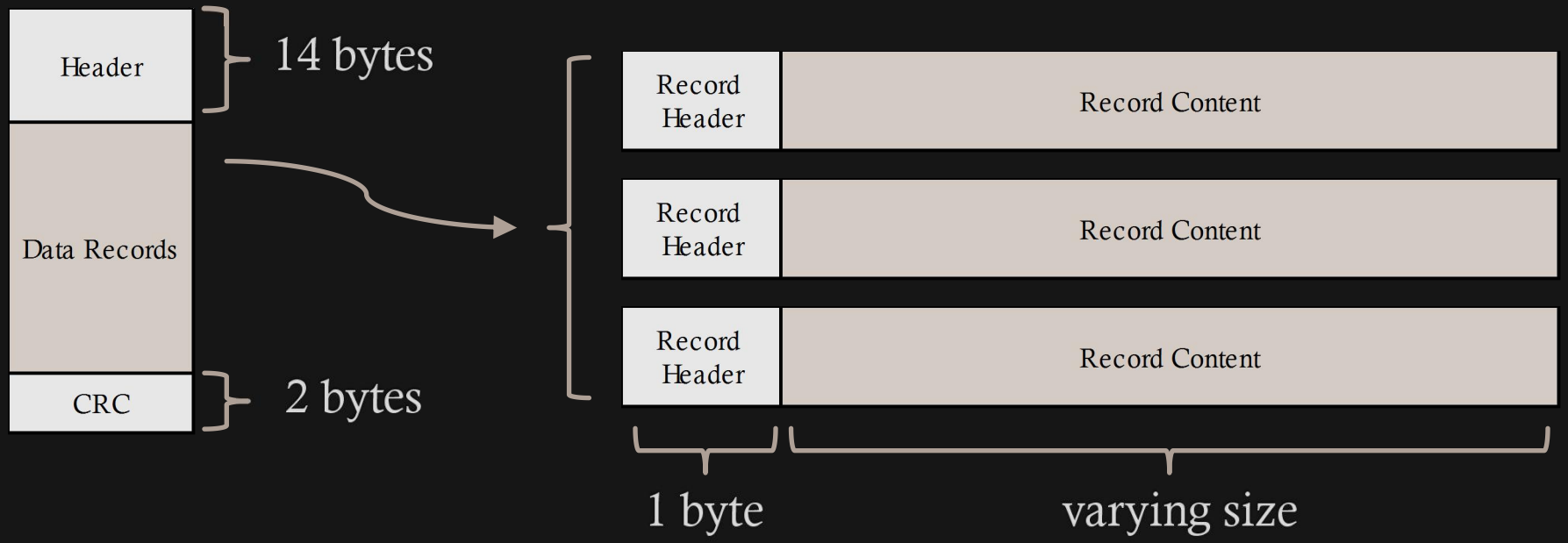
let those insects carry docs! (ANT-FS)

- SYNC device profile
- collect and transfer sensor data
- storage format: FIT protocol
- data transfer: ANT-FS

observe my mad Paint.NET skillz! :D

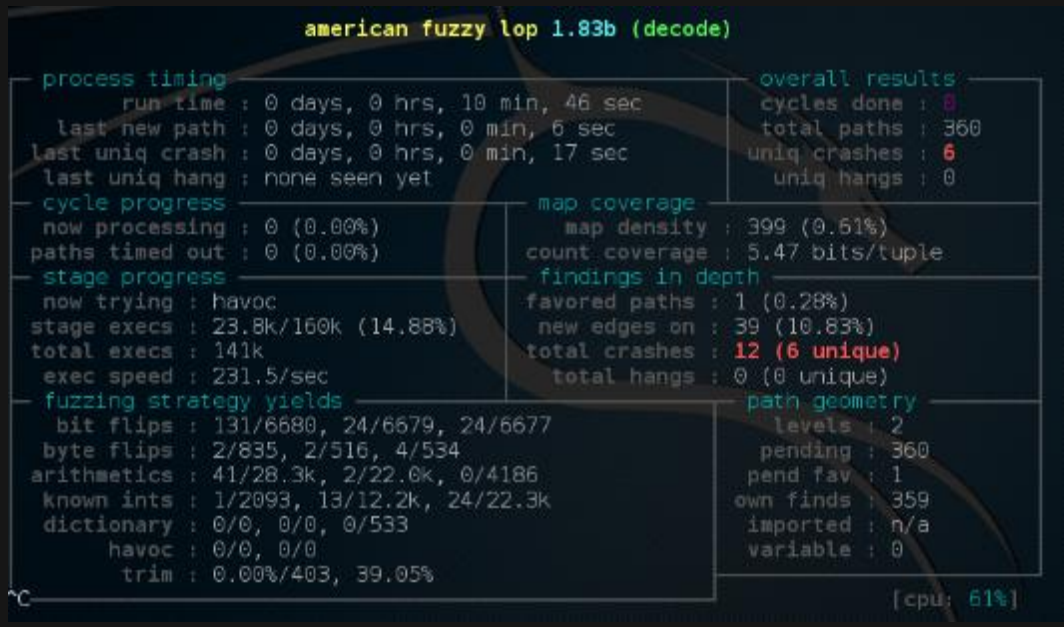
Flexible and Interoperable Data Transfer

- basically the file format for ANT-FS
- header + multiple records:



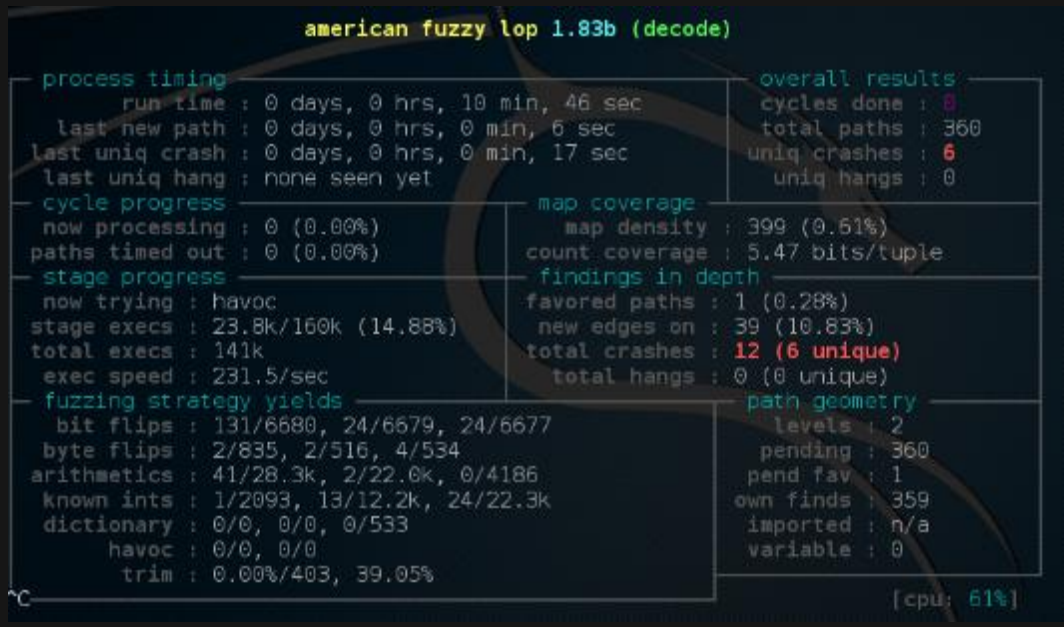
Flexible and Interoperable Data Transfer

- first attempt to hack my Garmin 310XT: FIT-fuzzing
- FIT SDK has some minimal samples
- gave the FIT decoder sample to everybody's favorite rabbit
- got some crashes :)
- sadly, seemingly non-exploitable ones



Flexible and Interoperable Data Transfer

- uploaded the crashing FIT files to the watch
- hoped for some crash dumps/logs, got nothing
- but the ANT stack became unavailable for a few minutes
- maybe integrated profile?
  - ANT chip handles FIT
  - if so, run code on that, instead of MCU?
  - future work…



… back to ANT-FS

- the sales pitch: a secure, reliable file transfer protocol built on ANT
- some rants on various forums might question the reliability
- I question the security
- two major security feature according to Dynastream:
  - built-in encryption – that nobody uses
  - multiple authentication mechanisms – we will see about them…

come on in, whoever you might be

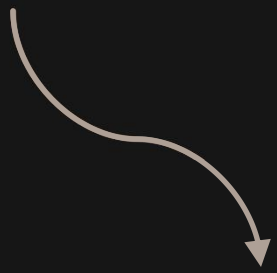
- a.k.a. the pass-through authentication mode
- not really authentication
- host asks nicely, client grants access
- the purpose?
  - testing stuff
  - maybe some legitimate uses
    e.g. partial track upload at bike race checkpoints

hey, do we let this guy in?

- a.k.a. pairing mode
- requires user interaction
- host sends serial number and friendly name
- client asks user about pairing
- if user accepts:
  - passkey is sent from client to host
  - host remembers passkey for future connections

attacks against pairing mode

- possible social engineering: send familiar friendly name and/or serial number
- passkey is sent on successful pairing



can be sniffed!!!

sniffing ANT

- nRF24AP1 and nRF24AP2 based on nRF24L01+
- 2.4 GHz ISM band
- 1 Mbps on-air data rate
- GFSK modulation
- actual packet format not clear – one of the documentations mentions Enhanced ShockBurst…

sniffing ANT with RTL-SDR

- based on http://blog.cyberexplorer.me/2014/01/sniffing-and-decoding-nrf24l01-and.html
- RTL-SDR + MMDS downconverter
- decoded rtl_fm output as Enhanced ShockBurst packe

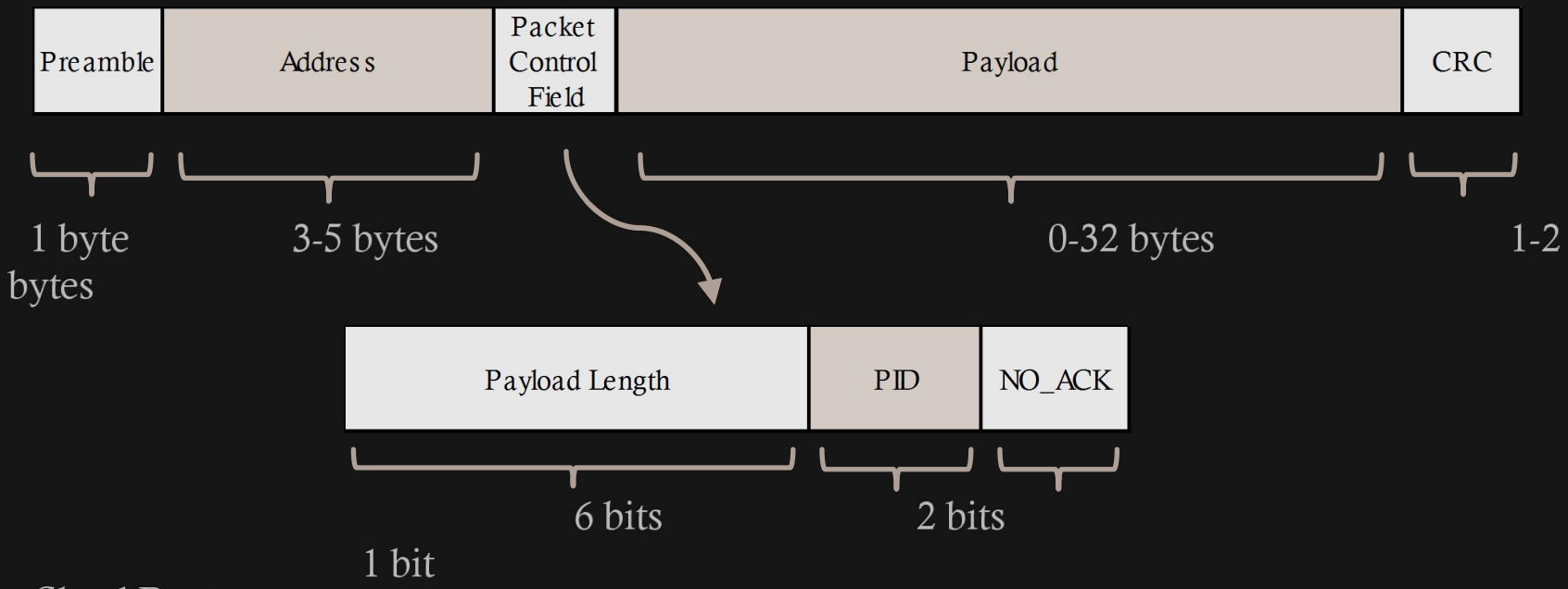
- **every byte was the double of the expected**

the mystery of the doubled values

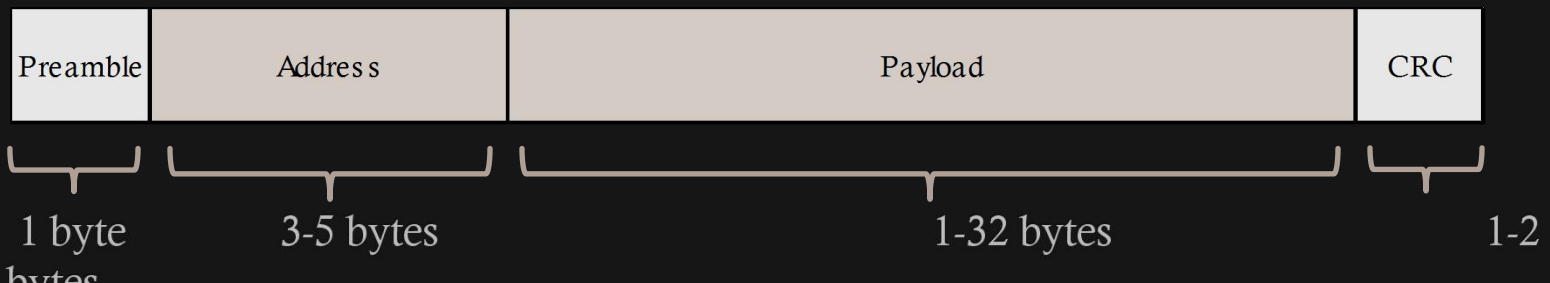
- they use a brand new, highly sophisticated encryption algo: **c = mul(m, k), where k = 2**
- would be a whole new level of dumb
- more possible reason: docs lied, packet format is not Enhanced ShockBurst

ShockBurst vs. Enhanced ShockBurst

Enhanced ShockBurst:



ShockBurst:



sniffing ANT-FS

- pairing Garmin 310XT with Garmin Express
- rtl_fm + shockburst.c + anteater.py
- Pothosware blocks (automatic frequency change on Link Command)

yeah sure, I know you, you can come in

- a.k.a. passkey mode
- passkeys are up to 255 bytes long
  - brute-forcing is impractical
- after successful pairing, host remembers passkey
- passkeys are stored per client serial
- if client with known serial found, host sends the corresponding passkey
- remember: everything is plaintext!

man-in-the-middle

man-in-the-middle

- using ANT-FS PC Tools to pose as host and client
- expected to find every info in debug logs
- passkey checking algo made it impossible:
  - passkey comes in 8-byte packets
  - the first packet contains the passkey length
  - if length is incorrect, authentication aborted
  - we don’t know the length :(

man-in-the-middle

- no problem, we can patch the tools
- specifically patch ANT_WrappedLib.dll
- plus point for Dynastream: the source is available
- no need for binary patching

man-in-the-middle

- attacker pose as an ANT-FS host, gets client's serial number
- attacker poses as client with obtained serial number, host sends her the passkey
- attacker connects to client with obtained passkey

possible timing attack

- passkey can be up to 255 bytes long
- checked in 8-byte chunks
- bail out on length mismatch
- bail out when wrong chunk received

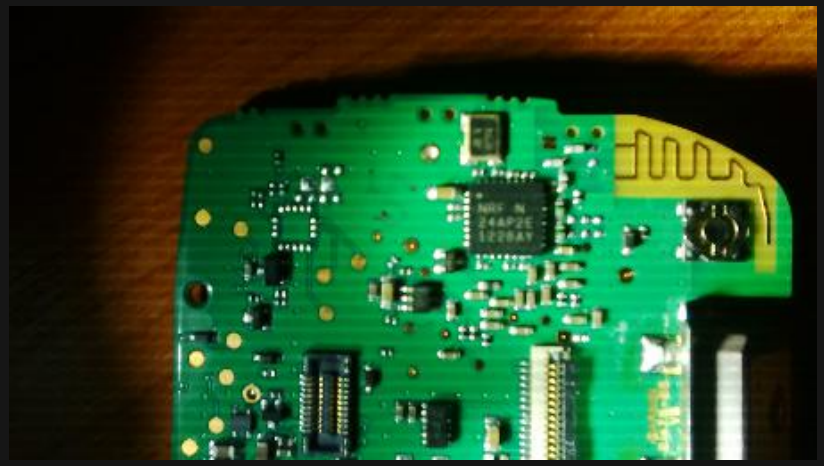
```
3511            }
3512            else
3513            {
3514               UCHAR ucCounter;
3515
3516               for (ucCounter = 0; ucCounter < 8; ucCounter++)
3517               {
3518                  if (ucPassKeyIndex >= ucPassKeySize)
3519                     break;
3520                  if (aucPassKey[ucPassKeyIndex++] != pucAuthCommand_[ucCounter])
3521                  {
3522                     bAcceptRequest = FALSE;  // Reject if passkeys are different
3523                  }
3524               }
3525            }
3526            if (ucControlByte_ & SEQUENCE_LAST_MESSAGE)   // last packet
3527            {
```

so, it's all crap?

- pretty much, yeah
- is it possible to use ANT securely? maybe:
  - don't use ANT+
  - purchase a private network key
  - use ANT chips with black/whitelist, inclusion/exclusion list, and encryption support

ANTs need no keys

- tried Garmin Forerunner 310XT with ANT-FS PC Host
- curiosity: what happens if I try pass-through auth?
- it worked. whose fault it is?
  - Garmin, or the ANT chip maker



ANTs need no keys

- authenticate to 310XT using pass-through
- download directory listing
- hackant.py (demo tool based on openant)

one key to ANT them all

- ANT-FS feature: OTA firmware updates
- some devices does not even have other means of data transfer
- the method is documented by Dynastream
- of course Garmin does not use that method
- reverse engineered the Garmin Express services to learn how it actually works

one key to ANT them all

- two methods caught my eye
  - ComputeFactoryPairedDevicePasskey
  - HasFactoryPairedDevicePasskey
- realization:
  - e.g. bundled heartrate monitor and sport watch
- it seems that it works for all Garmin gadgets
- added it to hackant.py

```
91      def factory_passkey(self, device_id):
92          n = ((device_id ^ 0x7d215abb) << 32) + (device_id ^ 0x42b93f06)
93          passkey = [(n >> (8 * i) & 255) for i in range(8)]
94          return passkey
```

one key to ANT them all

- try connect my Vivovit with a random key (it fails)
- try with the generated factory key (it succeeds)

one key to ANT them all

- serial number can be obtained by posing as host
- or read from the device / packaging

who signs firmware anyways?

- already mentioned OTA firmware updates
- it works like this with Garmin stuff:
  - updates are .rgn files, that have to be unwrapped
  - first file in ANT-FS directory is the firmware
  - update is a simple overwrite as “DEVICE” file type

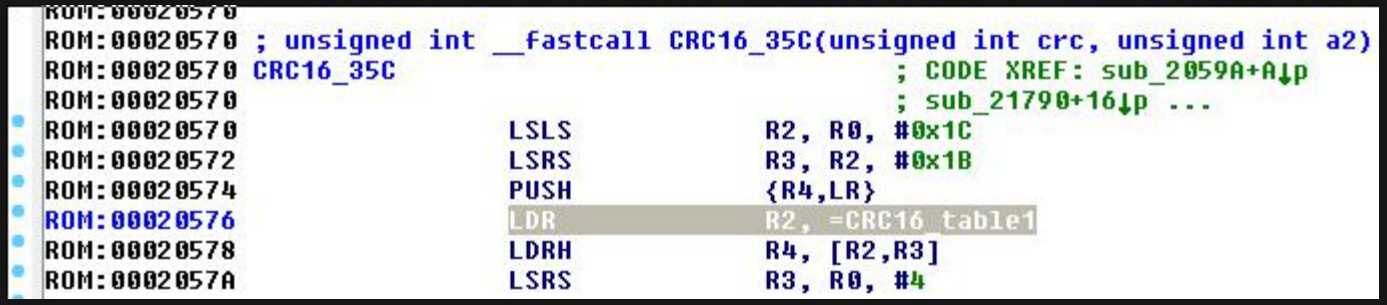
who signs firmware anyways?

- can't upload modified firmware. checksum?
- found two CRC16 tables, and two CRC16 functions

```
[0x00000000]> /x 000001cc
Searching 4 bytes in [0x0-0xf4f4]
hits: 2
0x0000efc0 hit0_0 000001cc
0x0000f010 hit0_1 000001cc
[0x00000000]> pf[16]w @hit0_0
0x0000efc0 = [ 0x0000, 0xcc01, 0xd801, 0x1400, 0xf001, 0x3c00, 0x2800, 0xe401, 0xa001, 0x6c00,
0x7800, 0xb401, 0x5000, 0x9c01, 0x8801, 0x4400 ]
[0x00000000]> pf[16]w @hit0_1
0x0000f010 = [ 0x0000, 0xcc01, 0xd801, 0x1400, 0xf001, 0x3c00, 0x2800, 0xe401, 0xa001, 0x6c00,
0x7800, 0xb401, 0x5000, 0x9c01, 0x8801, 0x4400 ]
[0x00000000]>
```

who signs firmware anyways?

- turned out they were not used for firmware check
- but they were useful!
  - functions used direct addresses to the tables
  - this made it possible to deduce fw load address



a nice example of how a failed attempt can yield useful info

who signs firmware anyways?

- actual CRC algorithm:
  sum of all bytes have to be zero
- last byte = -1 * sum(previous bytes)

```
 1 bool IsFirmwareOK_sub_84A()
 2 {
 3   char v0; // r0@1
 4
 5   v0 = 1;
 6   if ( (unsigned int)v20002402 - 1 < 0xFBFF )
 7     v0 = SumBytes_sub_82D8((_BYTE *)0x2FC00, v20002402);
 8   v20002402 = 0;
 9   return v0;
10 }
```

```
 1 int __fastcall SumBytes_sub_82D8(_BYTE *bytes, unsigned int length)
 2 {
 3   _BYTE *_bytes; // r3@1
 4   int result; // r0@1
 5   unsigned int i; // r2@1
 6
 7   _bytes = bytes;
 8   result = 0;
 9   for ( i = 0; i < length; ++i )
10     result += _bytes[i];
11   return result;
12 }
```

who signs firmware?

- unwrap RGN using firmware.py
- modify firmware
- calculate CRC with fixcrc.py
- upload modified firmware using hackant.py

really-really love XML

- XML string in Vivofit firmware
- two nodes with suspicious values: "XXXXXXXXXX" and "X"
- found a function that replaces these with
  - Device ID
  - a value based on whether HR monitor is used or not

some kind of device descriptor file

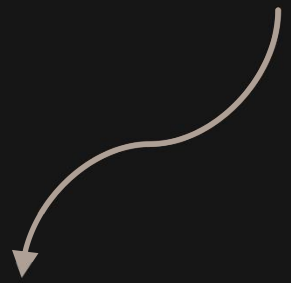
really-really love XML

- ANT-FS deals with FIT files
- Vivofit also BTLE-capable – uses it to connect to Garmin Connect
- found functions in Garmin Connect that download this XML

```
2409    @Override
2410    public void readGarminDeviceXml(long l, RemoteFileTransferCallback remoteFileTransferCallback)
2411        throws RemoteException {
2412
2413        Parcel parcel = Parcel.obtain();
2414        Parcel parcel2 = Parcel.obtain();
2415        try {
2416            parcel.writeInterfaceToken("com.garmin.android.deviceinterface.RemoteGdiService");
2417            parcel.writeLong(l);
2418            IBinder iBinder = remoteFileTransferCallback != null ?
2419                remoteFileTransferCallback.asBinder() : null;
2420            parcel.writeStrongBinder(iBinder);
2421            this.mRemote.transact(26, parcel, parcel2, 0);
2422            parcel2.readException();
2423            return;
2424        }
2425        finally {
2426            parcel2.recycle();
2427            parcel.recycle();
2428        }
2429    }
```

really-really love XML

- ANT-FS deals with FIT files
- Vivofit also BTLE-capable – uses it to connect to Garmin Connect
- found functions in Garmin Connect that download this XML



- a few years ago reported some XXE to Garmin (no response, BTW)
- so maybe Connect is XXE-able too?

XXE-ing Garmin

- replace Vivofit's XML with XXE stub …
- wait for connection from my phone
- connection comes, just not from the phone, but from a Garmin server!!!
- not the most usual way to find these kind of bugs :)

summary

- ANT is bad, m'kay?
- easy to sniff and MitM
- security measures are not really security measures
- it is possible to steal files,
- or even update firmware over the air, without user knowledge
- Garmin firmware are not signed – easy to modify

contact

- name: Tamas Szakaly
- mail: tamas.szakaly@praudit.hu
  sghctoma@gmail.com
- PGP fingerprint:
  4E1F 5E17 7A73 2C29 229A CD0B 4F2D 6CD0 9039 2984
- GitHub: https://github.com/sghctoma
- twitter: @sghctoma

links & credits

- ANT resources- https://www.thisisant.com
- FireChat - http://opengarden.com/firechat
- The Internet of Insecure Things - https://courses.csail.mit.edu/6.857/2016/files/camelosa-greene-loving-otgonbaatar.pdf
- Sniffing and decoding NRF24L01+ and Bluetooth LE packets for under $30 - http://blog.cyberexplorer.me/2014/01/sniffing-and-decoding-nrf24l01-and.html
- Garmin Express - http://software.garmin.com/en-US/express.html
- Garmin Connect - https://play.google.com/store/apps/details?id=com.garmin.android.apps.connectmobile&hl=en