Designing RF Fuzzing Tools to Expose PHY Layer Vulnerabilities

Matt Knight

GNU Radio Conference 2018

GNU Radio
THE FREE & OPEN SOFTWARE RADIO ECOSYSTEM

River Loop Security

whois

Matt Knight

- Senior Security Engineer at Cruise Automation
- RF Principal at River Loop Security
- BE in EE from Dartmouth College
- Software, hardware, and RF engineer
- RF, SDR, and embedded systems

Ryan Speers (here in spirit)

- Co-founder at River Loop Security
- Computer Science from Dartmouth College
- Cryptography, embedded systems, IEEE 802.15.4, automated firmware analysis

River Loop Security

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, & David Dowd
- Idiosyncrasies in PHY implementations

Mechanisms for automating:

- RF fuzzing
- Bug discovery
- PHY FSM fingerprint generation

Agenda

1. Overview of traditional fuzzing techniques (software and networks)
   > How these do and don't easily map to RF
2. RF fuzzing overview and state of the art
3. Ideal fuzzer design
4. TumbleRF introduction and overview
5. TumbleRF usage example

Traditional Fuzzing Techniques

What is fuzzing?

Measured application of pseudorandom input to a system

Why fuzz?

- Automates discovery of crashes, corner cases, bugs, etc.
- Unexpected input → unexpected state

What can one fuzz?

Fuzzers generally attach to system interfaces, namely I/O:

- File format parsers
- Network interfaces
- Shared memory

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?

Other Applications of Fuzzing

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
- ...

Fuzzing RF

WiFuzz

- MAC-focused 802.11 protocol fuzzer

Marc Newlin's 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

RF fuzzing projects are siloed / protocol-specific

- COTS radio chipsets
- Generally limited to MAC layer and up

RF state is hard to instrument

- What constitutes a crash / bug / etc?

Layer 2 implies 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

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

- `0xXXXX0000 == 802.15.4 Preamble`
- `0xAF       == 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?

Systematic Discovery via Fuzzing

Ideal RF Fuzzer Design

Ideal Features

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

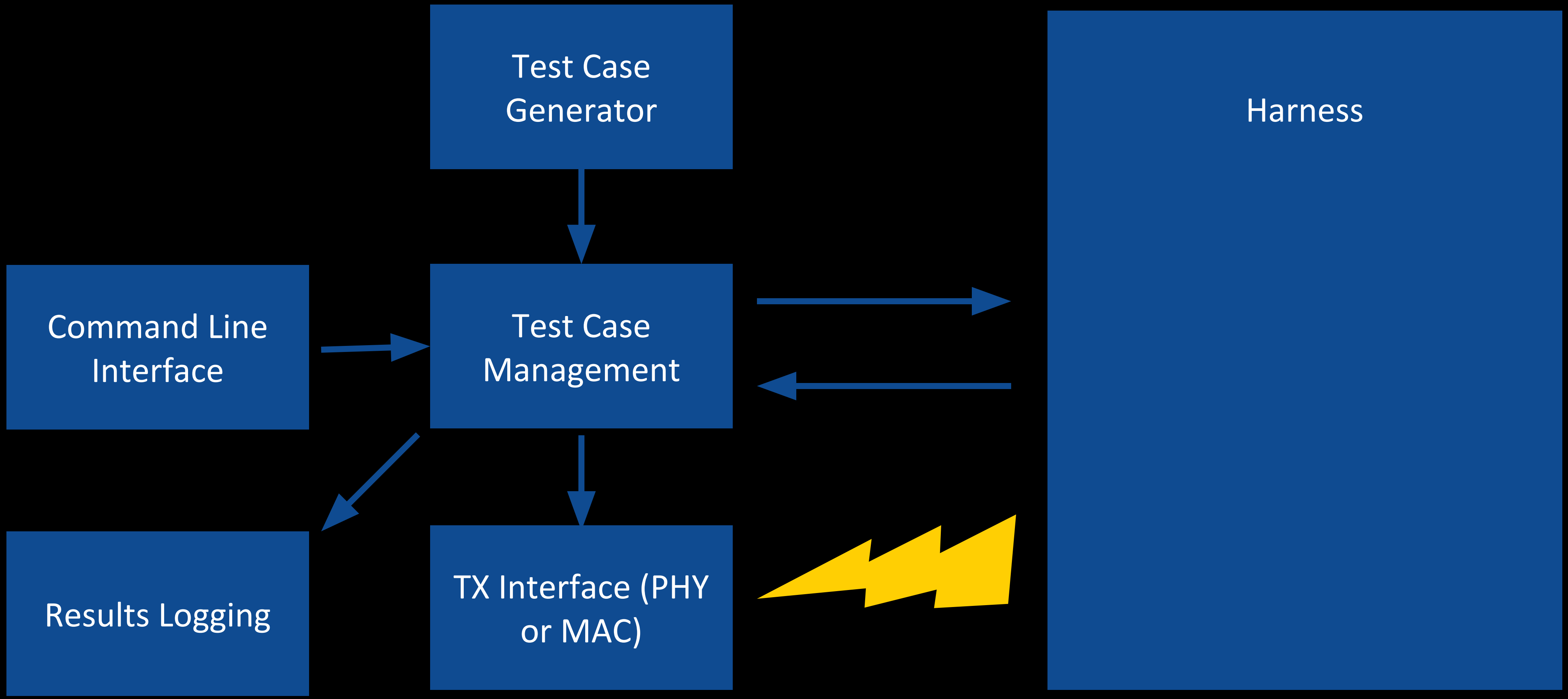
TumbleRF

Software framework enabling fuzzing arbitrary RF protocols

Abstracts key components for easy extension:

- Radio API
- Test case generation API
- Harness API

TumbleRF Architecture



Interfaces

RF injection/sniffing functions abstracted to generic template

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

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

TODO: [set/get]_symbol_rate()
```

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 output 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 check for crashes / ensure interface is still up

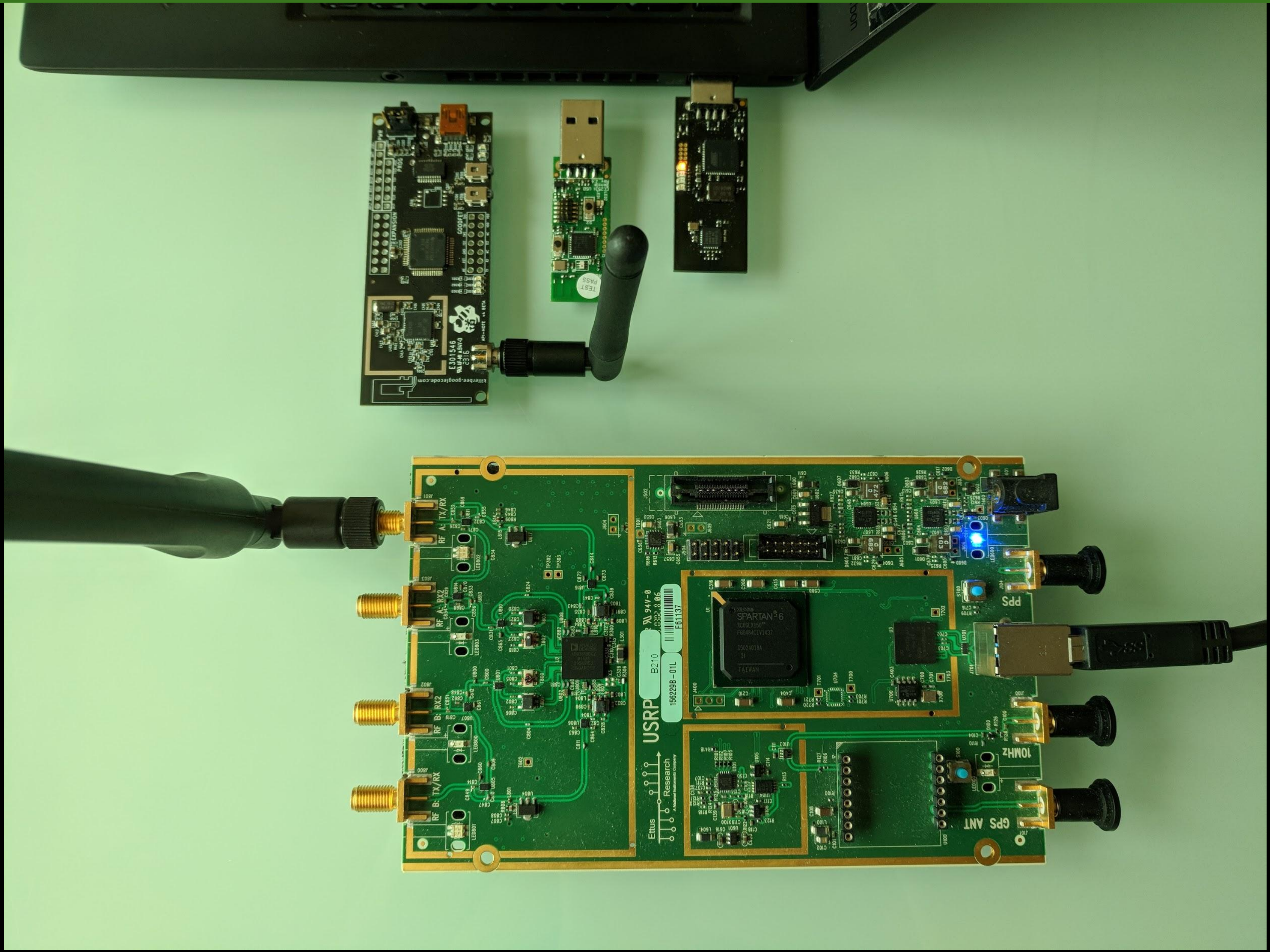
Test Setup (1/2)

Test Setup (2/2)

Devices Under Test

(left to right)

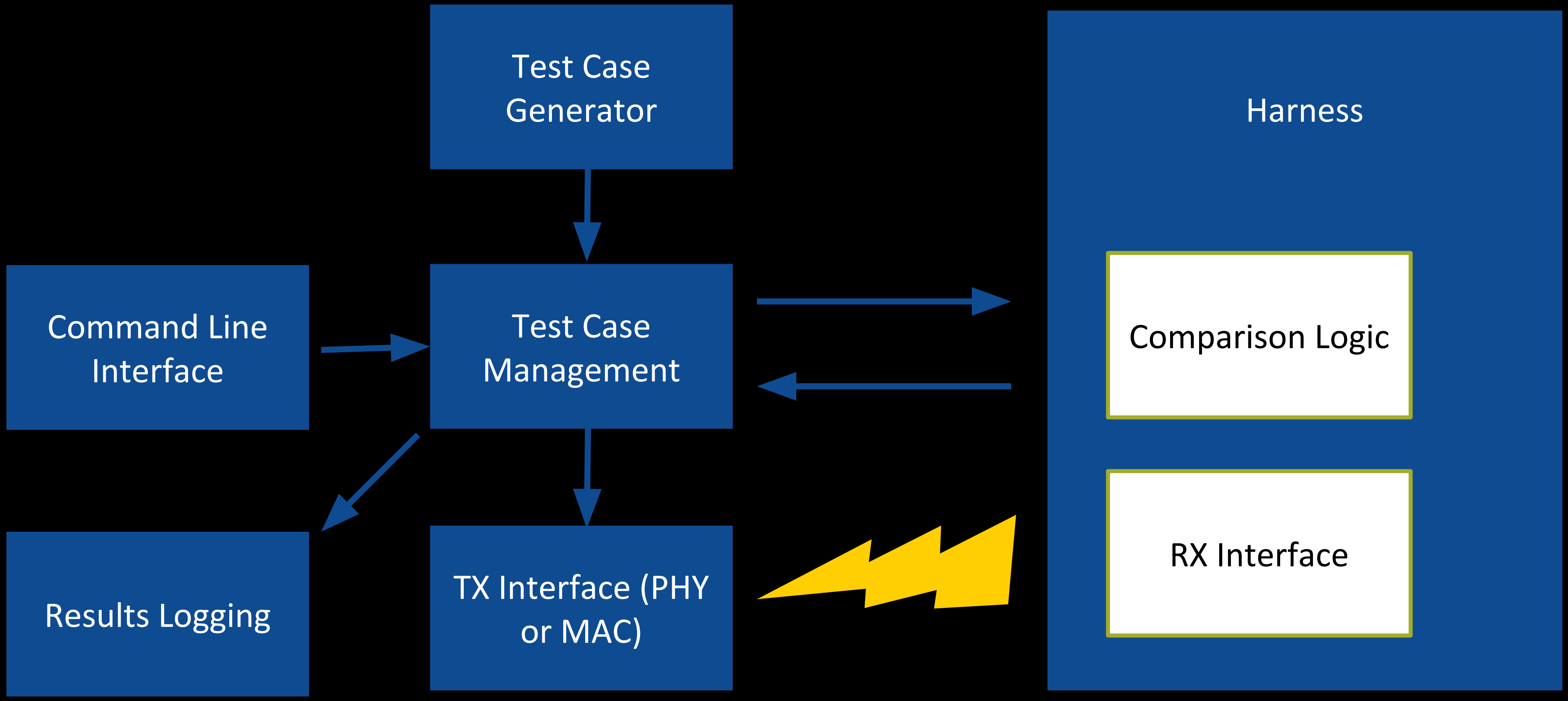
- TI CC2420
- TI CC2531
- Atmel AT86RF230



Stimulus

- USRP B210

TumbleRF Architecture: Demo Setup



Generated Data: Preamble Length

Standard 802.15.4 PHY Header == 0x00000000 + 0xA7 + 0xLL

Preamble				SFD	Length
0x00	0x00	0x00	0x00	0xA7	0xLL

Generated Data: Preamble Length

Standard 802.15.4 PHY Header == 0x00000000 + 0xA7 + 0xLL

Preamble				SFD	Length
	0x00	0x00	0x00	0xA7	0xLL

Generated Data: Preamble Length

Standard 802.15.4 PHY Header == 0x00000000 + 0xA7 + 0xLL

Preamble				SFD	Length
		0x00	0x00	0xA7	0xLL

Generated Data: Preamble Length

Standard 802.15.4 PHY Header == 0x00000000 + 0xA7 + 0xLL

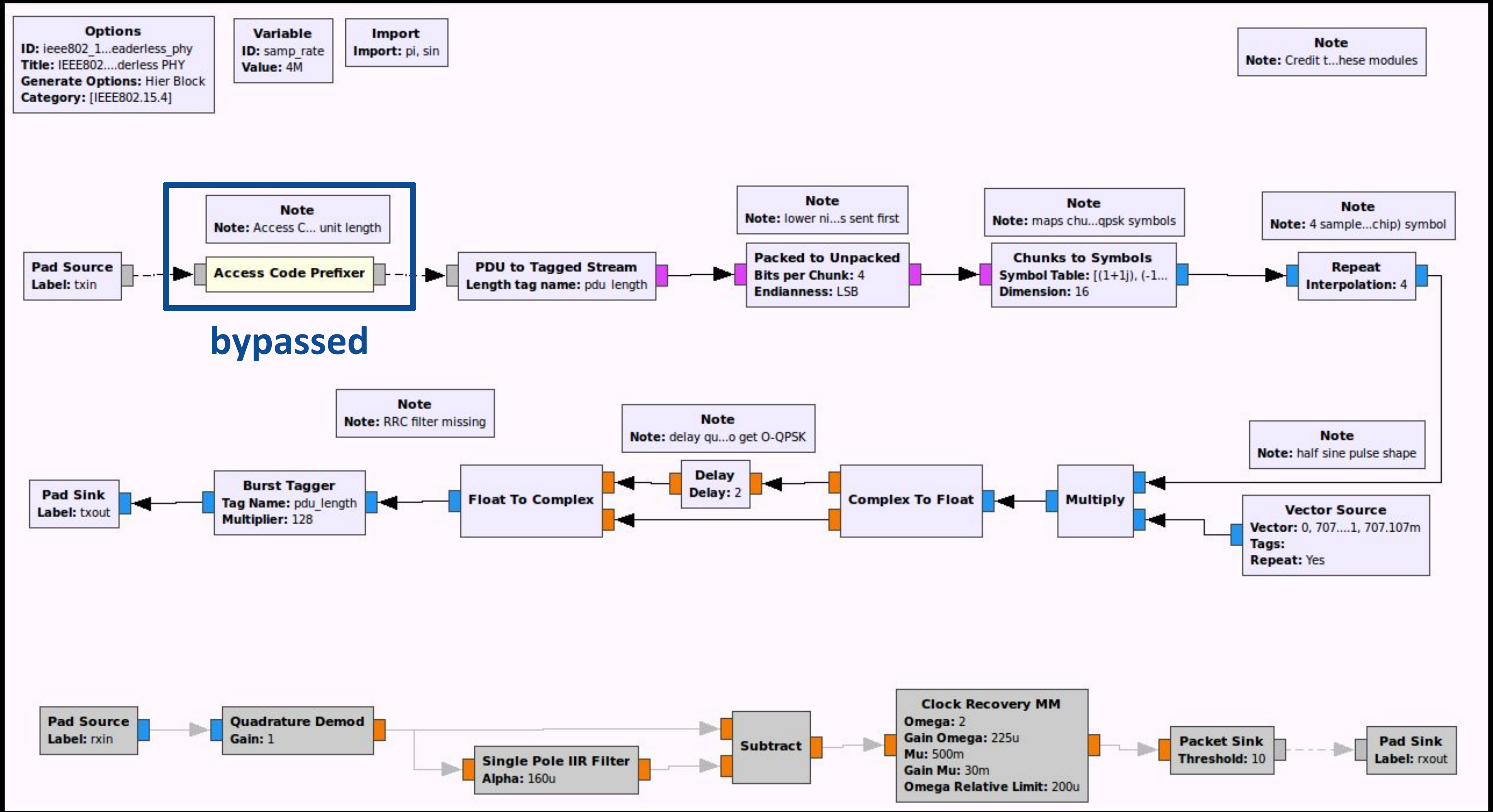
Preamble				SFD	Length
			0x00	0xA7	0xLL

Generated Data: Preamble Length

Standard 802.15.4 PHY Header == 0x00000000 + 0xA7 + 0xLL

Preamble				SFD	Length
				0xA7	0xLL

Generated Data: Preamble Length



Modify GNU Radio *gr-ieee802-15-4* to omit PHY header

Generate arbitrary PHY headers via TumbleRF test case generator

Demo

Results Dump

TI CC2420

Test: preamble_length_**apimote**.json (using Dot15d4PreambleLengthGenerator)

```
Case 0: 0 valid, 50 invalid     example case: a70a230800ffff000007fba6
Case 1: 0 valid, 50 invalid     example case: 70aa308220f0ff0f0070d0eafa
Case 2: 45 valid, 5 invalid     example case: 00a70a230804ffff00000757b6
Case 3: 0 valid, 50 invalid     example case: 0070aa308260f0ff0f007010e0fb
Case 4: 50 valid, 0 invalid     example case: 0000a70a230808ffff000007a387
Case 5: 0 valid, 50 invalid     example case: 000070aa3082a0f0ff0f007050fff8
Case 6: 50 valid, 0 invalid     example case: 000000a70a23080cffff0000070f97
Case 7: 0 valid, 50 invalid     example case: 00000070aa3082e0f0ff0f007090f5f9
Case 8: 48 valid, 2 invalid     example case: 00000000a70a230810ffff0000074be4
Case 9: 0 valid, 50 invalid     example case: 0000000070aa308220f1ff0f0070d0c1fe
```

Atmel AT86RF230

Test: preamble_length_**rzusbstick**.json (using Dot15d4PreambleLengthGenerator)

```
Case 0: 0 valid, 50 invalid     example case: a70a230800ffff000007fb
Case 1: 0 valid, 50 invalid     example case: 70aa308230f0ff0f007060
Case 2: 0 valid, 50 invalid     example case: 00a70a230805ffff000007
Case 3: 0 valid, 50 invalid     example case: 0070aa308270f0ff0f0070
Case 4: 0 valid, 50 invalid     example case: 0000a70a230809ffff0000
Case 5: 0 valid, 50 invalid     example case: 000070aa3082b0f0ff0f00
Case 6: 37 valid, 13 invalid    example case: 000000a70a23080effff00
Case 7: 0 valid, 50 invalid     example case: 00000070aa308200f1ff0f
Case 8: 41 valid, 9 invalid     example case: 00000000a70a230813ffff
Case 9: 0 valid, 50 invalid     example case: 0000000070aa308250f1ff
```

TI CC2531

Test: preamble_length_**cc2531**.json (using Dot15d4PreambleLengthGenerator)

```
Case 0: 0 valid, 50 invalid     example case: a70a230800ffff000007fba6
Case 1: 0 valid, 50 invalid     example case: 70aa308220f0ff0f0070d0eafa
Case 2: 13 valid, 37 invalid    example case: 00a70a230804ffff00000757b6
Case 3: 0 valid, 50 invalid     example case: 0070aa308260f0ff0f007010e0fb
Case 4: 48 valid, 2 invalid     example case: 0000a70a230808ffff000007a387
Case 5: 0 valid, 50 invalid     example case: 000070aa3082a0f0ff0f007050fff8
Case 6: 50 valid, 0 invalid     example case: 000000a70a23080cffff0000070f97
Case 7: 0 valid, 50 invalid     example case: 00000070aa3082e0f0ff0f007090f5f9
Case 8: 49 valid, 1 invalid     example case: 00000000a70a230810ffff0000074be4
Case 9: 0 valid, 50 invalid     example case: 0000000070aa308220f1ff0f0070d0c1fe
```

3 transceivers
2 manufacturers
1 protocol

3 behaviors!

Why Care?

Those results can allow for WIDS evasion and selective targeting.

Interested? Get Involved!

Contribute something to TumbleRF:

- Radio interface to fuzz your favorite protocol
- Generator for some cool new fuzzing idea you have
- Harness to check the state of a device you care about testing

Investigate other use cases:

- Test orchestration
- Applications to hardware interfaces/other types of PHYs

https://github.com/riverloopsec/tumblerf

Thank You!

GNU Radio Conference 2018

River Loop Security
Cruise Automation

GNURadio
THE FREE & OPEN SOFTWARE RADIO ECOSYSTEM

River Loop Security

https://github.com/riverloopsec/tumblerf

Questions?

@embeddedsec

matt@riverloopsecurity.com

GNURadio
THE FREE & OPEN SOFTWARE RADIO ECOSYSTEM

River Loop Security