Designing RF Fuzzing Tools to Expose PHY Layer Vulnerabilities

Matt Knight
GNU Radio Conference 2018
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
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
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
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
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
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
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

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

Short preamble?

strategically malformed
Sync Words and Magic Numbers

Turns out not all sync words are created equally

- \(0xXXXX0000\) == 802.15.4 Preamble
- \(0xA\text{F}\) == 802.15.4 Sync Word

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

- Test Case Generator
  - Test Case Management
  - TX Interface (PHY or MAC)
  - Results Logging
  - Command Line Interface

Harness
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:

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

TODO: [set/get]_symbol_rate()
Generators

Rule sets for generating fuzzed input (pythonically)
Extend to interface with software fuzzers of your choice

Implement 2 functions:
- \texttt{yield\_control\_case()}
- \texttt{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 (2/2)

Devices Under Test
(Left to right)

- TI CC2420
- TI CC2531
- Atmel AT86RF230

Stimulus

- USRP B210
TumbleRF Architecture: Demo Setup

Test Case Generator

TX Interface (PHY or MAC)

Test Case Management

Command Line Interface

Results Logging

Harness

Comparison Logic

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

<table>
<thead>
<tr>
<th>Preamble</th>
<th>SFD</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00</td>
<td>0x00</td>
<td>0x00</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0xA7</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0xLL</td>
</tr>
</tbody>
</table>
Generated Data: Preamble Length

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

<table>
<thead>
<tr>
<th>Preamble</th>
<th>SFD</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00</td>
<td>0x00</td>
<td>0x00</td>
</tr>
<tr>
<td></td>
<td>0xA7</td>
<td>0xLL</td>
</tr>
</tbody>
</table>
Standard 802.15.4 PHY Header == 0x00000000 + 0xA7 + 0xLL

<table>
<thead>
<tr>
<th>Preamble</th>
<th>SFD</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00</td>
<td>0x00</td>
<td>0xA7</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0xLL</td>
</tr>
</tbody>
</table>
Standard 802.15.4 PHY Header == 0x00000000 + 0xA7 + 0xLL

<table>
<thead>
<tr>
<th>Preamble</th>
<th>SFD</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0x00</td>
<td>0xLL</td>
</tr>
</tbody>
</table>
Standard 802.15.4 PHY Header == 0x00000000 + 0xA7 + 0xLL

<table>
<thead>
<tr>
<th>Preamble</th>
<th>SFD</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0xA7</td>
<td>0xLL</td>
</tr>
</tbody>
</table>
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)

<table>
<thead>
<tr>
<th>Case</th>
<th>Valid</th>
<th>Invalid</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>50</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>50</td>
</tr>
<tr>
<td>2</td>
<td>45</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>50</td>
</tr>
<tr>
<td>4</td>
<td>50</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>50</td>
</tr>
<tr>
<td>6</td>
<td>50</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>50</td>
</tr>
<tr>
<td>8</td>
<td>48</td>
<td>2</td>
</tr>
<tr>
<td>9</td>
<td>0</td>
<td>50</td>
</tr>
</tbody>
</table>

Example cases:
- Case 0: a70a230800fff0000007fba6
- Case 1: 70aa308220f0ff00070d0eafa
- Case 2: 00a70a230804ffff000000757b6
- Case 3: 0070aa308260f0ff0007010e0fb
- Case 4: 0000a70a230808ffff0000007a387
- Case 5: 000070aa3082a0ff0007050f0f8
- Case 6: 000000a70a230810ffff00000074be4
- Case 9: 0000000070aa308220f0ff00070d0c1fe

TI CC2531

Test: preamble_length_cc2531.json (using Dot15d4PreambleLengthGenerator)

<table>
<thead>
<tr>
<th>Case</th>
<th>Valid</th>
<th>Invalid</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>50</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>50</td>
</tr>
<tr>
<td>2</td>
<td>13</td>
<td>37</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>50</td>
</tr>
<tr>
<td>4</td>
<td>48</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>50</td>
</tr>
<tr>
<td>6</td>
<td>50</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>50</td>
</tr>
<tr>
<td>8</td>
<td>49</td>
<td>1</td>
</tr>
<tr>
<td>9</td>
<td>0</td>
<td>50</td>
</tr>
</tbody>
</table>

Example cases:
- Case 0: a70a230800fff0000007fba6
- Case 1: 70aa308220f0ff00070d0eafa
- Case 2: 00a70a230804ffff000000757b6
- Case 3: 0070aa308260f0ff0007010e0fb
- Case 4: 0000a70a230808ffff0000007a387
- Case 5: 000070aa3082a0ff0007050f0f8
- Case 6: 000000a70a230810ffff00000074be4
- Case 9: 0000000070aa308220f0ff00070d0c1fe

Atmel AT86RF230

Test: preamble_length_rzusbstick.json (using Dot15d4PreambleLengthGenerator)

<table>
<thead>
<tr>
<th>Case</th>
<th>Valid</th>
<th>Invalid</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>50</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>50</td>
</tr>
<tr>
<td>2</td>
<td>45</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>50</td>
</tr>
<tr>
<td>4</td>
<td>50</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>50</td>
</tr>
<tr>
<td>6</td>
<td>50</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>50</td>
</tr>
<tr>
<td>8</td>
<td>48</td>
<td>2</td>
</tr>
<tr>
<td>9</td>
<td>0</td>
<td>50</td>
</tr>
</tbody>
</table>

Example cases:
- Case 0: a70a230800fff0000007fba6
- Case 1: 70aa308220f0ff00070d0eafa
- Case 2: 00a70a230804ffff000000757b6
- Case 3: 0070aa308260f0ff0007010e0fb
- Case 4: 0000a70a230808ffff0000007a387
- Case 5: 000070aa3082a0ff0007050f0f8
- Case 6: 000000a70a230810ffff00000074be4
- Case 9: 0000000070aa308220f0ff00070d0c1fe

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
Thank You!

GNU Radio Conference 2018

River Loop Security
Cruise Automation

https://github.com/riverloopsec/tumblerf