Phantom Boundaries and Cross-layer Illusions in 802.15.4 Digital Radio

(Research Report)

Travis Goodspeed
Straw Hat Security
travis@radiantmachines.com

Abstract*—The classic design of protocol stacks, where each layer of the stack receives and unwraps the payload of the next layer, implies that each layer has a parser that accepts Protocol Data Units and extracts the intended Service Data Units from them. The PHY layer plays a special role, because it must create frames, i.e., original PDUs, from a stream of bits or symbols. An important property implicitly expected from these parsers is that SDUs are passed to the next layer only if the encapsulating PDUs from all previous layers were received *exactly as transmitted by the sender and were syntactically correct.

The Packet-in-packet attack (WOOT 2011) showed that this false assumption could be easily violated and exploited on IEEE 802.15.4 and similar PHY layers; however, it did not challenge the assumption that symbols and bytes recognized by the receiver were as transmitted by the sender. This work shows that even that assumption is wrong: in fact, a valid received frame may share no symbols with the sent one! This property is due to a particular choice of low-level chip encoding of 802.15.4, which enables the attacker to co-opt the receiver's error correction. This case study demonstrates that PHY layer logic is as susceptible to the input language manipulation attacks as other layers, or perhaps more so. Consequently, when designing protocol stacks, language-theoretic considerations must be taken into account from the very bottom of the PHY layer; no layer is too low to be considered "mere engineering."

***Keywords*-LangSec, 802.15.4, Packet-in-packet**

I. Background

Key property of network stacks: The key property of network stacks is that the objects constructed by the receiving stack at any layer are equivalent to those transmitted by the sending stack. Platform differences like endianness and message loss due to noise notwithstanding, the assumption that the contents of successfully received messages are exactly the same as of those transmitted is so fundamental that it hardly ever gets explicitly specified as a security requirement. Yet this implicitly assumed property deserves a closer look.

Each layer of the OSI stack model deals with its particular kind of object: Physical layer (PHY) with encoded continuous signals, Link layer (LNK) with frames, Network layer with packets, and so on for Transport, and, where implemented, for Session and Presentation layers. Thus the primary function of a receiving layer is to *produce respective objects for the next layer*, in a manner faithful to the intent and content of the sender's transmissions.

Functioning of layers is commonly understood and described in terms of *addressing*: e.g., LNK frames are addressed to specific nodes or broadcast to groups of nodes on a local link, Network packets are uniquely addressable to nodes in a global network, Transport layer specifies receiving applications on a (globally addressed) node, and so on. Formats and fields of these layers' data structures are explained in terms of such addressing. However, this view obscures the key fact that each layer in the receiving stack serves as first and foremost a constructor of specific data objects from a serialized form in which they exist in the SDU.

Layers as transducers: The key fact of the OSI stack and similar network stack designs is that each layer is actually a *transducer* for received data, consisting of a *recognizer* for the intended objects' serial presentation (which may be a stream of encoded bits, symbols, bytes, or tokens) and a *constructor* of the resulting objects to pass to the next layer (provided that recognition succeeded). As is common with input-handling code at communication boundaries, recognizer false positives lead to constructor code being handed data it doesn't expect, which in turn leads to computation driven by such data taking unexpected paths, that is to say, exploitation. This paper, however, focuses on code never leaving expected paths, but we still achieve those delightfully counter-intuitive results that make security worth studying.

The recognizer's first task is to detect the boundaries of such representations in the stream and to accumulate them for parsing. The PHY layer handles this problem in its most pure form, being tasked with distinguishing between the out-of-frame state in which it judges received bits or symbols to be line/ether noise and the in-frame state in which it judges and records the incoming data as contents of a frame.

Unlike PHY, the higher layers aren't tasked with discarding "noise" or "garbage" bytes, but must still find boundaries of their respective objects such as protocol headers in their PDUs. The key—though often implicit—security property of the stack implementation is that these boundaries must be matched exactly as meant and encoded by the sender. In presence of noise, this is a non-trivial computational task,

A short version of this paper, describing a practical, hands-on bypass of the protection technique in [1], appeared in [2]. This paper gives a theoretical LangSec perspective on the original attack, the bypass, and the challenges of designing defensible PHY layers.

DOI 10.1109/SPW.2014.33

which requires an appropriate automaton with that specific property.

We previously ([3]) demonstrated a non-obvious failure of 802.15.4 PHY recognizer: the simple form of the *Packet-in-packet* attack that works by including the symbols that make up a Physical layer frame in the payload of Application layer. Normally, the interior bytes of a packet are escaped by the outer frame's header, but collisions sometimes destroy that header. However, these collisions tend to be short and often leave the interior of the packet intact, damaging only the beginning or the end. On a busy band like 2.4GHz, this happens often enough that it can be used reliably to inject frames into a remote network, without owning a radio.

Our description of the Packet-in-packet attack prompted mitigations such as [1]. However, these transmitter-side mitigations took for granted that the symbols recognized by the receiver—i.e., nybbles of bytes passed from the PHY to the LNK layer—were exactly as transmitted by the sender. In this paper we show that this assumption is incorrect, and that the design of 802.15.4 PHY actually allows a received frame to share *no symbols at all* with the transmitted one.

In this paper we undertake a case study of one digital radio stack, 802.15.4. Let the reader not mistake the specific character of this case study for a lack of breadth: many other PHY layers follow the same design and are therefore susceptible to the surprises we describe. We argue that stack security and correctness is a formal language recognition problem even at the lowest layers of PHY, and that no layer or sublayer design can hope to escape them and remain defensible.

II. The Layer Cake is a PHY!

The PHY layer is often described as dealing with "raw bits", which it makes into frames. This presumes at least one recognizer automaton that decides which raw bits belong in a frame and which don't. However, even a cursory look at PHY shows that there are in fact several layers and representations of bits, and therefore not one but many automata.

In particular, the 802.15.4 PHY layer encodes its symbols, corresponding to the nybbles of the PHY frame, as sequences of 32 *chips*, the "native" ones and zeros of the protocol's modulation. Thus "raw bits" of the frame are certainly not raw; rather, they are extracted from a stream of chips by an appropriate automaton.

Are "raw bit" boundaries real?: All object boundaries in higher protocol layers ultimately depend on the symbol boundaries as constructed on the PHY layer. When even the bits of a frame are themselves so constructed, correctness of all the other layer's interpretation of a message hinges on the constructing automaton—including the above-mentioned fundamental property that any successfully received object was transmitted just so by the sender.

0	—	11011001110000110101001000101110
1	—	11101101100111000011010100100010
2	—	00101110110110011100001101010010
3	—	00100010111011011001110000110101
4	—	01010010001011101101100111000011
5	—	00110101001000101110110110011100
6	—	11000011010100100010111011011001
7	—	10011100001101010010001011101101
8	—	10001100100101100000011101111011
9	—	10111000110010010110000001110111
A	—	01111011100011001001011000000111
B	—	01110111101110001100100101100000
C	—	00000111011110111000110010010110
D	—	01100000011101111011100011001001
E	—	10010110000001110111101110001100
F	—	11001001011000000111011110111000

Figure 1. 802.15.4 symbols, encoded in chips.

Everything about how we represent frames and packets in writing—or even in coding—seems to reinforce the idea that nybble boundaries are "natural" and "hard." We think of noise-induced receiver errors as flipping bits, but we implicitly assume that it's the bits, not their boundaries, that get corrupted. Even this corruption is conveniently swept under the rug, due to checksums.

In fact, all such boundaries are imaginary, and even frame boundaries detection (SFD matching) is dependent on a lower layer of chip-level error correction.

Error correction: The chip sequences specified by the 802.15.4 standard to represent symbols (as in Fig. 1) are an error connecting code that gets processed transparently to the rest of the stack. This processing happens in a layer of its own, before SFD matching and frame construction, and is not normally a part of security modeling.

Thus the standard four zero-byte preamble of an 802.15.4 frame is actually eight repetitions of the zero-symbol `11011001110000110101001000101110` *or any sufficiently similar sequence of chips*, and similarly with the `A7` Start-of-Frame-Delimiter and the rest of the frame. Thus any received frame has many potential chip-level representations that will be received equivalently. Not only that, but these representations also need to be *extracted from the continuous stream of chips* received by the PHY radio.

Crucially, this stream does not honor any hard boundaries between the contiguous chip groups that would be matched as nybbles. Such boundaries are merely an abstraction, a product of interpretation by the receiver—performed, at this level, by error correcting logic. Can this logic be manipulated by the sender to produce an unexpected, boundary abstraction-busting result in the receiver?

The eighth-of-a-nybble misalignment trick: Consider the chip code listings in Fig. 1 and apply a rotation by $\frac{1}{8}$ of their length to each. This rotation brings each valid code into another valid code, and the codes form two "rings" under such rotation. In other words, this rotation maps the set of codes into itself, and its repeated action separates it into two orbits (formally speaking, these are the two orbits of the cyclic group's $\mathbb{Z}_8$ action on the set of codes, but we won't be using this formalism).

Thinking geometrically, this property is merely a symmetry of the set of 802.15.4's chosen code points in the hypercube $\{0, 1\}^{32}$. Such symmetries are expected of optimal error correcting codes that place their points at the largest possible Hamming distance from each other. Assuming that errors randomly flip the chips of a transmitted code, which is then mapped to the closest code point, largest-distance placement helps the code survive the largest number of chip-flip errors without actually producing a wrong symbol.

However, code chips are received as a stream rather than as separate words with enforced boundaries; the boundaries are merely an abstraction. This, shifting the chip stream by "an eighth of a nybble"—that is, changing the receiver's idea of the stream's start or timing—will produce a valid sequence of symbols! For example, a zero shifted once will produce 1 (or 7 if the shift is in the opposite direction), while the same zero shifted twice will produce 2 (or 6), and so on.

```
0                               11011001110000110101001000101110
1                           11101101100111000011010100100010
2                       00101110110110011100001101010010
3                   00100010111011011001110000110101
4               01010010001011101101100111000011
5           00110101001000101110110110011100
6       11000011010100100010111011011001
7   10011100001101010010001011101101

8                               10001100100101100000011101111011
9                           10111000110010010110000001110111
A                       01111011100011001001011000000111
B                   01110111101110001100100101100000
C               00000111011110111000110010010110
D           01100000011101111011100011001001
E       10010110000001110111101110001100
F   11001001011000000111011110111000
```

In the more convenient hexadecimal notation (but remembering that each nybble is actually a 32-bit chip sequence):

```
0        D9C3522E
1       ED9C3522
2      2ED9C352
3     22ED9C35
4    522ED9C3
5   3522ED9C
6  C3522ED9
7 9C3522ED

8        8C96077B
9       B8C96077
A      7B8C9607
B     77B8C960
C    077B8C96
D   6077B8C9
E  96077B8C
F C96077B8
```

Receiving a frame that was never transmitted: The stream misalignment trick described above allows a sender to craft a frame that would cause a *different* frame to be received by a standard-compliant receiver. The received frame would, in fact, share *no symbols at all* with the transmitted one.

Furthermore, such transmission can be accomplished on an fully standards-compliant sender. Access to the transmitting digital radio's configuration registers to change the transmitted SFD (such as is provided by the CC2420 digital radio IC) would simplify matters, but is not required, since the sender can at worst use the Packet-in-packet technique and rely on noise for successful injection, as detailed in [3].

Let us now send a frame using nothing but misaligned symbols. The frame needs to start with the standard preamble and SFD; by the time we figure out how to represent them, the principle for crafting the rest of the frame will be clear.

First, consider sending a preamble of eight 0 symbols. At the chip sublayer of PHY, we have, in the shorthand notation above:

```
    0        0        0        0        0        0        0        0
D9C3522E D9C3522E D9C3522E D9C3522E D9C3522E D9C3522E D9C3522E D9C3522E
```

Instead of 0-symbols, we will send 1-symbols, as follows. In this new crafted sequence, the central part is exactly correct, with sub-symbol errors occurring only at the edges. Note that these errors never exceed $\frac{1}{8}$ of a symbol.

```
    0        0        0        0        0        0        0        0
 D9C3522E D9C3522E D9C3522E D9C3522E D9C3522E D9C3522E D9C3522E D9C3522E
   1        1        1        1        1        1        1        1
ED9C3522 ED9C3522 ED9C3522 ED9C3522 ED9C3522 ED9C3522 ED9C3522 ED9C3522
```

Similarly, considering the opposite rotation, we can send `77777777`, incurring just as much error as above: at most 4 chips.

Next, we must follow up the preamble with the Start of Frame Delimiter, (`A7`). Instead of sending `00000000A7`, we can send `11111111B0` or `7777777796`.

Would the receiver be fooled? Dealing in actuality with the stream of chips and having no idea where the actual intended symbol boundaries are, the receiver must match arriving chip sequences with the standard code points; it cannot produce any other output than a defined symbol (based on matching against the Hamming-closest code-point)—no matter what the input sample!

So when the "in-frame" recognizer expects *A7* and receives *B0*, the first chip sequence `7B8C960`*D* instead of the `7B8C9607` is only off by the last four chips, with a Hamming distance of merely 2:

```
BO —— 77B8C960D9C3522E
         ||||||||
A7 ——  7B8C96079C3522ED
```

Continuing into the frame body, we can keep the Hamming distance between the misaligned crafted frame and the receiver's idea of it modulo error correction at or below 4. This Hamming distance is obviously *less* than that of the intended error correction distance of the 802.15.4 code.

Thus we can craft entire sender frames, complete with their preambles and SFDs, that have no symbols in common with the received frame!

III. Conclusions and Future Directions

Our 802.15.4 case study shows that the lowest layers of PHY are susceptible to unintended interpretations of data. By now we are used to such unintended interpretations in higher layers, such as various kinds of SQL injection (SQLi), command injection, and, more generally, "in-band signaling" vulnerabilities in application protocols. However, on the byte level there is little conceptual difference between Packet-in-packet and SQLi, as they are caused by similar mismatches between the downstream code's expectations of input-processing logic and the algorithmic reality of this logic (cf. [4], which applied language-theoretic insights to input validation, using SQLi as an example). Computationally simple recognizers cannot discern the intent of bytes or symbols in the input stream, yet subsequent code is written as if they could.

Why We Need LangSec Stacks: We see this case study as evidence that no part of the stack that transforms inputs can be left to "mere engineering," be it ever so classic and venerable. Wherever there is recognition and interpretation of data, formal language-theoretic principles of recognizing input must underlie that layer's or sublayer's design.[5]

Specifically, any structures in the input must be extracted and the validity of the entire input decided only by a well-defined computation model, such as a finite state or pushdown automaton, and the validity of input data must be specified in terms of a formal language matching that model, such as a regular grammar or a context-free grammar, with simplest possible language preferred. The alternative is confusion between convenient higher layer abstractions such as intended meanings or even boundaries of input bytes, symbols or bits, and the actual properties of the recognizers that are expected to faithfully recreate these abstractions while altogether lacking the computational power to do so.

Since data recognition and transforming action is central to network stacks no matter what other functions (such as encapsulation, addressing, or routing) guide their design, we envision secure stacks designed around strict language recognition disciplines: *LangSec Stacks*.

A New Hope?: A promising new direction to avoid the confusion between data and signaling from the PHY layer up has been presented by Michael Ossmann and Dominic Spill in [6], [7]. They present an Isolated Complementary Binary Linear Block Code (ICBLBC) encoding scheme that uses two distinct, separated error correcting codes for data and signaling info, and spaces the points of these codes to avoid confusion due to error. While designed explicitly to avoid traditional packet-in-packet attacks, these codes also resist our new variant of that technique for receiver manipulation, and may provide a way forward for lower sublayers of PHY.

Not only digital radio: Finally, we note that just as Packet-in-packet attacks aren't limited to digital radio—as demonstrated by Barisani et al. in [8]—the above PHY manipulation technique is likely not so limited either.

Acknowledgments

We would like to thank Julien Vanegue for pointing out the role of transducers in unexpected computation models (weird machines) and Felix 'FX' Lindner for the discussion of a LangSec Stack as a principle for future networking architectures. Thanks are also due to Sergey Bratus for his stubborn—and valuable!—refusal to stop harping about fingerprinting.

References

[1] A. Biswas, A. Alkhalid, T. Kunz, and C.-H. Lung, "A Lightweight Defence against the Packet in Packet Attack in ZigBee Networks," Wireless Days (WD), IFIP, November 2012.

[2] T. Goodspeed, "An Advanced Mitigation Bypass for Packet-in-Packet; or, I'm burning 0day to use the phrase 'eighth of a nybble' in print," *International Journal of PoC——GTFO*, vol. 3, no. 5, March 2014.

[3] T. Goodspeed, S. Bratus, R. Melgares, R. Shapiro, and R. Speers, "Packets in Packets: Orson Welles' In-Band Signaling Attacks for Modern Radios," in *5th USENIX Workshop on Offensive Technologies*, D. Brumley and M. Zalewski, Eds. USENIX, August 2011.

[4] R. J. Hansen and M. L. Patterson, "Guns and Butter: Towards Formal Axioms of Input Validation," Black Hat USA, August 2005, http://www.blackhat.com/presentations/bh-usa-05/BH_US_05-Hansen-Patterson/HP2005.pdf.

[5] L. Sassaman, M. L. Patterson, S. Bratus, and M. E. Locasto, "Security Applications of Formal Language Theory," *IEEE Systems Journal*, vol. 7, no. 3, pp. 489–500, 2013, http://langsec.org/.

[6] M. Ossmann and D. Spill, "Unambiguous Encapsulation - Separating Data and Signaling," Great Scott Gadgets Technical Report 2014-03-1, March 2014, http://greatscottgadgets.com/tr/gsg-tr-2014-1.txt.

[7] D. Spill and M. Ossmann, "Unambiguous Encapsulation: Separating Data and Signaling," ShmooCon 2014, January 2014, https://archive.org/details/ShmooCon2014_Unambiguous_Encapsulation.

[8] A. Barisani and D. Bianco, "Fully Arbitrary 802.3 Packet Injection: Maximizing the Ethernet Attack Surface," BlackHat USA, August 2013, https://media.blackhat.com/us-13/US-13-Barisani-Fully-Arbitrary-802-3-Packet-Injection-WP.pdf.