A Lightweight Defence against the Packet in Packet Attack in ZigBee Networks

Anshuman Biswas, Abdulaziz Alkhalid, Thomas Kunz, Chung-Horng Lung
Department of Systems and Computer Engineering, Carleton University
Ottawa, Ontario, K1S 5B6, Canada
{anshuman, akhalid, tkunz, chlung}@sce.carleton.ca

Abstract—**ZigBee is a standard for low-power, low-rate wireless communication. It is modeled on the physical layer and medium access control defined in IEEE 802.15.4. In this paper, we study an attack on modern radios called the "Packet in Packet" (PIP) attack. We replicate the attack in ZigBee devices and discuss possible defenses against the attack. We implement a solution to thwart the PIP attack in TinyOS and show through extensive experiments that the solution defends against the attack, while imposing little additional overhead.**

Keywords—ZigBee; TinyOS; Wireless; Communications; KillerBee; Security; Defence; Packet-in-Packet Injection

I. Introduction

ZigBee is a low-power, low data rate wireless protocol with a throughput of up to 250 kbps [2, 3]. It uses a small, lightweight stack (120 KB). However, where simplicity and low cost are goals, security frequently suffers. The Packet in Packet (PIP) attack is an in-band signaling mechanism that can be abused to inject raw digital frames, given control of data in the interior of a frame [1]. The mechanism depends on injecting a raw frame into the Layer 1 from upper-layer protocols through abusing in-band signaling methods. In-band signaling [4] allows signaling data to be transferred in the same packet as user data by piggybacking the signaling data. The piggybacking technique allows any attacker to hide malicious packets inside a normal packet payload that is permitted onto the network. When the Packet-in-Packets (PIPs) traverse through the wireless network, a bit error in the outer frame causes the inner frame to be interpreted instead of the outer frame. This enables the attacker to evade the firewalls and intrusion detection/prevention systems, user-land networking restrictions and similar defenses. Using the interior fields of higher networking layers, the packet will be constructed. Thus, the attacker only needs authority to send a clear text data over the air even if it is wrapped within several network layers. The motivation of our work is to study the PIP attack technique in ZigBee networks and propose and evaluate a solution based on byte stuffing to counter the attack with little overhead. Using an approach based on byte stuffing, we are able to thwart the attack, with little overhead in terms of network throughput or packet latency.

The rest of this paper is organized as the following: Section II demonstrates the PIP attack on TinyOS motes. Section III presents our chosen defence, based on byte stuffing, discusses our implementation and presents experimental results. Section IV discusses our conclusions.

II. Demonstrating the PIP Attack

We used TelosB devices in our experiments. TelosB is an open source, low-power wireless sensor module designed by MEMSIC. We exploit the attack as a characteristic of the PHY layer. Using ZigBee makes it easier to demonstrate the attack as it does not implement certain features implemented in other wireless technologies like IEEE 802.11. The environment was setup in an Ubuntu 12.04, 64 bit box. We cross-compiled the GoodFET [5] firmware. We also installed the KillerBee [6] framework on our machine to compile GoodGET. Or scenario had Trudy send malicious packets to Bob routed via Alice. Once our TelosB nodes with the appropriate firmware were flashed and connected to the machine, we configured one to act as the sniffer node and the others to act as Alice and Trudy. In our scenario, Trudy would send her packets to Bob via Alice. The sniffer node was acting as Bob and picked up all packets directed to it. To check whether the attack was working, we confirmed that a node (even a sniffer) could mistake a data inside the payload to be a valid frame.

```
#goodfet.ccspi sniff 1 head
Listening as 00deadbeef on 2405
#DEBUG Clearing Overflow
#2f 01 08 82 de ff ff ff ff de ad be ef ba be c0 00 00 00 00 a7 0f 01 08
82 ff ff ff ff de ad be ef ba be c0 ff ff ff
#00 00 00 00 a7 0f 01 08 82 ff ff ff ff de ad be ef ba be c0 ff 1e
#2f 01 08 82 de ff ff ff ff de ad be ef ba be c0 00 00 00 00 a7 0f 01 08
82 ff ff ff ff de ad be ef ba be c0 ff ff ff
#2f 01 08 82 de ff ff ff ff de ad be ef ba be c0 00 00 00 00 a7 0f 01 08
82 ff ff ff ff de ad be ef ba be c0 ff ff ff
```

Figure 1. The result from the GOODFET sniffer

We realized that the number of packets with the outer frame missing increased considerably by reducing the transmission power, the sending frequency, and the distance from the receiver. We set the frequency to 2405 MHz, the distance was about 8m.

Figure 1 demonstrates the results. To sniff raw IEEE 802.15.4 packets on Channel 1, we used the sniff option in the goodfet.ccspi. We programmed Trudy to send a packet to Alice every 20ms, which she would forward to Bob. The output of the sniffer was the packet being sent from Alice to Bob. The sniffer did not suspect anything as the packet seemed to arrive from a legitimate source. The highlighted line in Figure 1 indicates where the packet injection was successful.

978-1-4673-4404-3/12/$31.00 ©2012 IEEE

In ZigBee, the symbol size is 1 nibble (4 bit), so we saw the damage of entire nibbles rather than individual bits. As long as the attack is aligned to a nibble boundary, the inner packet had a chance of being (mis-)interpreted. However, we noticed that when the malicious packet was inserted at a position not divisible by 4, the attack failed.

III. DEFENCE – BYTE STUFFING

In this section, we present our solution which we chose after investigating different alternatives. The proposed solution is based on bit stuffing. Bit stuffing is an error detection mechanism to stop control information from appearing in the payload of a MAC frame. As explained in [7], the beginning and end of frames are marked in HDLC by a unique 8-bit pattern (01111110) called a flag, without any explicit frame length field. To avoid the simulation of a flag within a frame, HDLC provides an escape mechanism called bit stuffing: the transmitter inserts a 0 (insertion 0-bit) after any 5 contiguous 1-bits in the frame, and the receiver deletes any 0-bit following 5 contiguous 1-bit (if there are more than 5 contiguous 1-bit this will be a flag, an abort or an idle sequence).

To implement that defense, we operate at the byte level instead of the bit level, as the smallest symbol used by ZigBee devices is at least one nibble long. Bit or byte stuffing does not suffer from the problems of key distribution as some of the other possible solutions we explored, using cryptographic defenses. Even if the attacker knows that byte stuffing is being performed at every node, there is no way to get around this defense, unless the attacker starts sending the packets directly to the victim, instead of routing it through an intermediate node. To compare between the performance of the non-byte stuffed implementation and the byte-stuffed solution, we designed two sets of experiments for measuring the throughput and the latency of the ZigBee nodes under similar scenarios. We used six TelosB motes, five of which were used for the experiment and the sixth one was dedicated to sniffing the packets in the air to measure the performance. To sniff packets graphically, we used a software called Z-Monitor [8].

We determine the throughput as the function of the number of contending nodes and the frequency of sending packets. For our experiment, we assume that four nodes are sending packets for 60 seconds to the fifth node, which after a certain interval sends a packet with the count of the number of packets it received within the 60 second window. Latency is measured by sending a packet that is returned to the sender and the round-trip time is considered the latency. We measure the ZigBee MAC latency by having a node inject packets every 3 seconds into a line network of multiple hops and wait for the time it takes for the packet to travel the network and return back to the originating node. We vary the number of hops by changing the number of nodes from 2 to 3 and 4. We have one node acting as the base station, which injects packets into the system. The last node reverses the direction of the packet, once it reaches it. The middle nodes forward the packet in a particular upstream or downstream direction, depending on the direction from which they received it. In both experiments, we keep a fixed packet size of 99 bytes for the payload and 14 bytes for the header. For the byte stuffed scenario, the on-air packet size may vary and depends on the number of byte stuffing operations performed, in addition to the original 99 bytes of payload.

We implemented byte stuffing as a middleware layer embedded in every node. In TinyOS, each node accesses the radio through a well-defined interface. We added an additional implementation of that interface which allowed the required functionalities of byte stuffing and de-stuffing operations to be performed. Hence the sender and receiver interface of a node were 'wired' to interact with our interface before communicating with the radio. From the application's perspective, nothing needs be changed other than the wiring of the application component to the radio. This meant that the solution was easy to deploy without requiring any changes to the existing code.

The middleware layer is deployed on all nodes in the byte stuffed scenario. This layer gets activated before the packet is sent through the radio. A reverse operation is performed every time a packet is received. The nodes scan through the payload to detect the occurrence of four continuous '00' symbols. Irrespective of whether the fifth symbol is a Sync 'A7' symbol, we byte stuff the payload with a 'FF'. On the other end, if the node encounters a set of four '00' symbols, followed by a 'FF' symbol; it removes the 'FF' symbol and continues reading the payload. Note that this also deals with byte patterns of for '00' symbols followed by an 'FF' in the original payload: the transmitted packet will have a sequence of four '00's followed by two 'FF's, one of which will be removed by the receiver. When the outer packet actually breaks, the injected inner packet will be misinterpreted as background noise as the recipient's radio would encounter an 'FF' symbol instead of the Sync symbol. We force the node to scan through the entire packet before forwarding it. It might be advisable to drop packets where the node is required to perform more than one byte stuffing operation. This may lead to the occasional packet getting lost, but it may lead to a significant improvement in the performance of our solution. Also, there exists a low probability of the same data packet containing two sets of four consecutive '00' symbols.

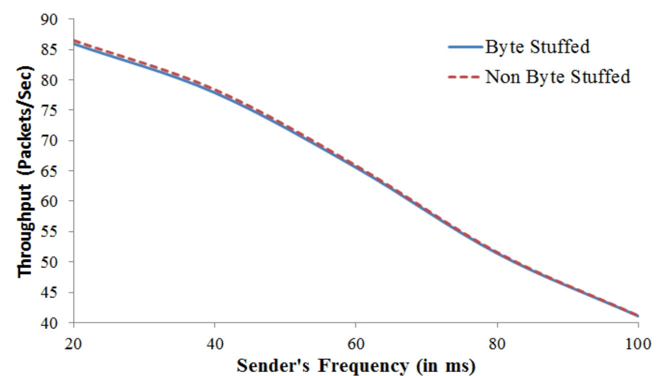

Figure 2. Throughput

In Figure 2, we compare the throughput of 4 ZigBee devices with and without byte stuffing. We have measured the

throughput of these devices at 5 different packet sending rates. This starts from 20ms and increments in a 20ms step size until 100ms. This means that the nodes start sending packets after the timer has elapsed for a particular frequency. The nodes are programmed to send packets to a receiver for 60 seconds, after which the receiver sends a packet to the sniffer with the count of the number of packets received during the 60 second period. In the byte stuffing scenario, each node has to perform the corresponding operations: the sender nodes ensures that the packets they send to the receiver have been processed by our byte stuffing routine. On the receivers end, the node must ensure that, before increasing the count of the packet received, it performed the de-stuffing operation and recovers the original payload. However, these operations show very little impact on the throughput. At lower intervals of 20 milli-seconds, we expect the nodes to generate higher overall throughput. Here, we observe a difference of 0.6% between the two scenarios. The byte stuffing solution performs almost as well as the non-byte stuffed ones at higher intervals (i.e., less traffic/offered load). Also, we observed that going below the 20 milliseconds timer value reduced the measured throughput. We expect that this happened as random access MAC protocols such as the one used in ZigBee have a saturation threshold. If the offered load exceeds this threshold, the channel capacity drops.

In Figure 3, we compare the latencies of the two scenarios. We changed the number of hops a packet has to travel. We have scenarios involving 2 nodes, 3 nodes, 4 nodes and finally 5 nodes, resulting in hop counts of 2 to 8. Each time the base node generates a packet; it sends it to its neighbor and awaits a reply. This packet travels through the line topology hop by hop. When the packet reaches the last node, the packet direction is reversed and is sent back to the base station. In the byte stuffing scenario, each node performs byte stuffing and de-stuffing and the base/initiator node sends a packet with a malicious packet inside its payload. The latency should increase linearly as a function of the hop count as the time to process the packet at each node should take nearly the same amount of time.

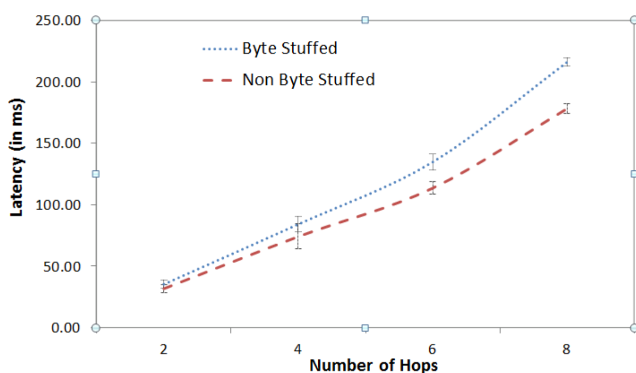

Figure 3. Latency

To calculate the latencies, we use 50 round trip times between the nodes and then calculate a 95% confidence interval. We observe from the graph that as the number of hops increases, the time spent in processing the packet increases as well, as expected. The increase is larger for byte stuffing. For 8 hops, we get a latency difference of around 38 milliseconds for the averages of the byte stuffing and non-byte stuffing scenarios. The confidence interval helps us evaluate the range overlap between the two scenarios. They show that starting with a 3 hop line topology the difference in performance becomes statistically significant at the 95% confidence level.

IV. CONCLUSIONS

The PIP attack demonstrates a way of affecting the network stack's perception of the lower layer medium and messages by merely manipulating the payloads of a higher layer. We have provided tested examples of using this technique, between networking layers of single radio architecture to inject raw frames between architectures that differ at the physical layer and to evade packet filtering defenses. Our results have demonstrated that an attacker can exploit this property wherever she is able to predict the on-air pattern produced by encapsulated data. This can be mitigated within a networking stack in a number of ways. We implemented a prevention based on byte stuffing, demonstrating that the defense is designed to thwart the PIP attack, and quantified the induced overheads. We observed that our solution has virtually no impact on the throughput. The latency of our solution does get affected significantly, as we use a packet size of 113 bytes and add extra bytes whenever we encounter the preamble in the payload. However, the latency increase is relatively small, accumulating to around 38 milliseconds for 8 hops or slightly less than 5 milliseconds per hop. This would also explain why we saw little or no change in the throughputs reported in Figure 2, as the additional latency is lower than the shortest packet interval.

REFERENCES

[1] T. Goodspeed, S. Bratus, R. Melgares, R. Shapiro and R. Speers, "Packets in Packets: Orson Welles' in-band signaling attacks for modern radios, "In Proceedings of the 5th USENIX conference on Offensive Technologies, 7-7, 2011

[2] D, Gascón, "Security in 802.15.4 and ZigBee networks," http://sensor-networks.org/index.php?page=0823123150, 2010

[3] Software Technologies Group, "How does ZigBee compare with other wireless standards?," http://www.stg.com/wireless/ZigBee_comp.html

[4] D. Preston, J. Preston and R. Leyendecke, "In-band signaling for data communications over digital wireless telecommunications network," US Patent, 2004

[5] T. Goodspeed, "GoodFET on the TelosB," http://travisgoodspeed.blogspot.ca/2011/03/goodfet-on-telosb-tmote-sky.html

[6] J. Wright, "KillerBee: Practical ZigBee Exploitation Framework," http://www.willhackforsushi.com/presentations/toorcon11-wright.pdf, October 2009

[7] D. Fiorini, M. Chiani, V. Tralli and C. Salati, "Can we trust in HDLC?" SIGCOMM Computer Communication Rev. 24, 5, pp. 61-80., 1994

[8] A. Koubaa, S. Chaudhry, O. Gaddour, R. Chaari,N. Al-Elaiwi, H. Al-Soli and H. Boujelben, "Z-Monitor: Monitoring and Analyzing IEEE 802.15.4-based