






# Implementation of a Wormhole Attack on Wireless Sensor Networks with XBee S2C Devices

Julian Ramirez Gómez<sup>(✉)</sup> ,  
Héctor Fernando Vargas Montoya<sup>(✉)</sup> ,  
and Alvaro Leon Henao<sup>(✉)</sup> 

Metropolitan Institute of Technology, Fraternidad Campus,  
050012 Medellín, Colombia  
julianbit.ramirez@gmail.com, hvargasm@gmail.com,  
alvarohenao7@gmail.com

**Abstract.** One of the most dangerous threats to Wireless Sensor Networks (WSN) are wormhole attacks due to their capacity to manipulate routing and application data in real time and cause important damages to the integrity, availability, and confidentiality of network data. In this work, an empirical method to launch such attack (which is successful) on IEEE 802.15.4/Zigbee devices with source routing enabled is adopted to find signatures for detecting wormhole attacks in real environments. It uses the KillerBee framework with algorithms for packet manipulation through a malicious node to capture and inject malicious packets in victim nodes. Besides, a reverse variant of wormhole attack is presented and executed. To evidence the realization of this threat by the attacking software, the experimental framework includes XBee S2C nodes. The results include recommendations, detection signatures and future work to face wormhole attacks involving source routing protocols like DSR.

**Keywords:** Wormhole · ZigBee · XBee · Source routing · Attack  
IoT · Cybersecurity · WSN · DSR · KillerBee

## 1 Introduction

A The Internet of Things (IoT) is a growing technology trend towards connecting all kinds of electronic devices to the Internet. The purpose of IoT devices is to interact and share information to ease end users' life. Thanks to it, by 2020 nearly 37 billion devices are going to be connected to the cyberspace [1]. Nevertheless, IoT is a new challenge in the information security field because a wide range of devices with different security features can be integrated, leading to a wider security gap. Furthermore, the implementation of security measures such as strong cipher protocols on devices with reduced processing power and memory, like environmental sensors, is a difficult task [2]. One of the most important IoT technologies are Wireless Sensors Networks (WSN), which can be deployed in many places (e.g. homes, buildings, cities, factories and hospitals) to monitor environmental variables: temperature, humidity, movement, lighting, and also to improve processes in the industrial field [3].

On the other hand, a considerable number of vulnerabilities and security threats related to WSNs has been presented in various research studies [4–6] that introduce potential damages to the integrity, availability, and confidentiality of the information in a WSN. Some of these threats are related to the network layer in the protocols stack. They include attacks selective forwarding, sinkholes, and wormholes, and are intended to induce an unwanted behavior in specific elements of WSNs through malicious nodes and traffic manipulation. These attacks are successful because they give an attacker the ability to intercept and modify data in real time, execute denials of service and selective forwarding attacks, store packets, inject false information into legitimate nodes and disrupt routing processes [7]. The risks of wormhole attacks represent new security gaps that must be addressed and reduced to protect end users' data and privacy.

### 1.1 Background

Wormhole attacks exploit the mechanisms to discover routes of on-demand routing protocols. The most remarkable cases are Ad-Hoc On-Demand Distance Vector (AODV) and Dynamic Source Routing (DSR) protocols, which use route request (RREQ) and route replay (RREP) packets as a way to discover routes by nodes in a WSN [8]. A RREQ packet is a broadcast message sent by a source node ("S") to request a route to a destination node ("D"), while an RREP is a unicast message sent by the destination node in response to an RREQ. Besides, when the RREP that contains the route to reach "D" arrives at "S", the source node stores the route collected by the RREP in the route cache and then sends the application data to "D" through that route. Accordingly, the main goal of wormhole attacks is to build a tunnel between two remote nodes through a third node ("M") placed within transmission range of "S" and "D". This occurs when "S" needs to send application data to "D" and broadcasts an RREQ message to discover a route to "D". "M" (which is listening to network traffic) forwards the message directly to "D" because the RREQ sent by "M" reaches "D" before the original RREQ through the direct link. "M" can listen to RREP from "D" first and then forward it to "S" with better metrics (zero hops), thus creating a false direct link between "S" and "D" through "M" in the process (Fig. 1).

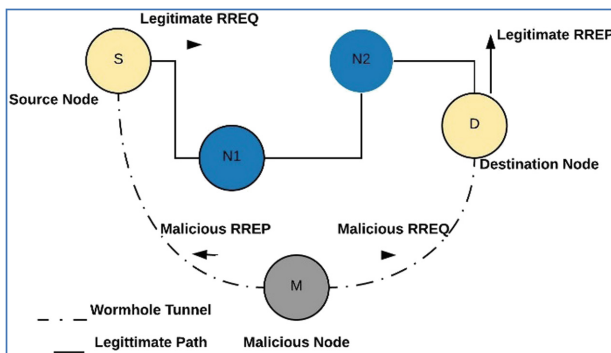


Fig. 1. Wormhole attack with malicious node

At this point, the attacker can control the data that flows through the malicious tunnel and launch other attacks. Finally, if victim nodes are too far from each other, the attacker can use two malicious nodes sharing a link to build the wormhole tunnel [9].

## 1.2 Related Work

In [10], wormhole attack detection is based on hop count and delay changes between source and destination nodes. If there is a wormhole tunnel between given source and destination nodes, the delay increases due to the longest path created by the wormhole tunnel, while hop count decreases for the same reason. In that sense, the detection scheme compares delay and hop count at a given moment with previous values to detect the attack. [11] proposes to apply modifications to the DSR routing protocol to automatically calculate a Round Trip Time (RTT) delay value between source and destination nodes at a given moment. Thus, initial RTT values are stored and compared with subsequent values of the same kind. If RTT changes, a wormhole attack is detected. Additionally, the network nodes are set in a promiscuous mode to monitor neighboring nodes. [12] introduces a modified version of the AODV routing protocol to calculate the transmission force from source nodes. The method aims to detect wormhole attacks with high transmission power by establishing a transmission power threshold for network nodes. If a node exceeds such threshold, it could be a compromised node and a wormhole attack is detected. In another modification of the AODV protocol [13], network nodes introduce the hash of the hop addresses and hop count into the RREQ packet while it follows a path from source to destination. When the RREQ packet reaches the destination node, the expected hash of RREP is calculated and compared with the received hash. If the hashes do not match, the packet is discarded assuming a wormhole attack in progress. In [10], to detect a wormhole attack, source nodes of RREQ calculate the delay between a sent RREQ and every received RREP to establish an average RTT value for all received routes. If the RTT of one or more routes is less than the average RTT, a wormhole attack is detected, malicious routes discarded, and the detection is replied to neighboring nodes to delete the malicious routes from their routing table. In [13], every node calculates changes in the number of neighboring nodes by counting neighbors at different times.

As a result, a wormhole attack is detected if a predefined threshold of the number of neighboring nodes is exceeded by one or more nodes. Besides, [14] presents a wormhole detection algorithm with node connectivity and statistical calculation. Such method defines two terms, node connectivity and network connectivity, to determine the probability of a wormhole attack in progress in the network. The probability of said attack depends on the network's density, which is based on the number of nodes and connections between nodes.

The research studies above conducted tests in simulation environments to measure the impact of wormhole attacks and the effectiveness of different detection/prevention algorithms in WSNs. Nevertheless, they are based on simulations of routing protocol attacks and are difficult to implement in real environments because of the lack of devices with the features required by the proposed methods. Due to existing and potential cybersecurity threats to WSNs, intrusion detection systems need to be developed for real sensor nodes. At last, since most WSN security research studies are

based on simulation results, future characterization of WSN threats should focus on real devices to build actual security solutions and prevent security disasters in WSN technologies.

To expose the flexibility of a wormhole attack and its impact on real cybersecurity environments, this paper proposes an algorithm to execute classic and “reverse” wormhole attacks on XBee S2C devices with source routing enabled. The main goal is to modify the route record field in routing packet headers to manipulate the routing cache in victim nodes. The algorithm is implemented in Python language using the KillerBee framework and an RZUSBSTICK dongle with preinstalled KillerBee firmware. The results include recommendations to prevent wormhole attacks, attack patterns and fingerprints to develop an Intrusion Detection System (IDS) for WSNs as future work.

## 2 Proposed Wormhole Attack Algorithm

The route record field in source routing packets [15, 16] contains the whole route from source to destination when the routing packet reaches the source of data transmission. This feature allows the intermediary hops between source and destination nodes to introduce their network address into the routing packets (RREP) while the packet follows the path from destination to source. A route is thus created and can be used by source nodes to send data packets to the corresponding destination of the source route, as shown in Fig. 2. When the route record field is void in received RREP packets, it means that both nodes source and destination are neighboring nodes.

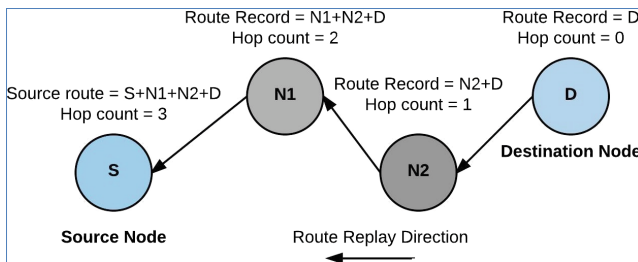


Fig. 2. Route record parameter process

In a classic wormhole attack, the main goal is to create a false neighborhood between two remote nodes through a third malicious node causing the route record field of RREP packets sent through the malicious links to be unmodifiable by intermediary nodes; as a result, they arrive at the destination with zero hops. This approach encompasses capturing packets, modifying the route record in RREP packets and injecting them into the source node to override its routing table with zero hop routes, which eventually builds a false neighborhood between source and destination nodes, as shown in Fig. 3. Consequently, the route record parameter needs to be modified because RREP packets could come from an intermediary node.

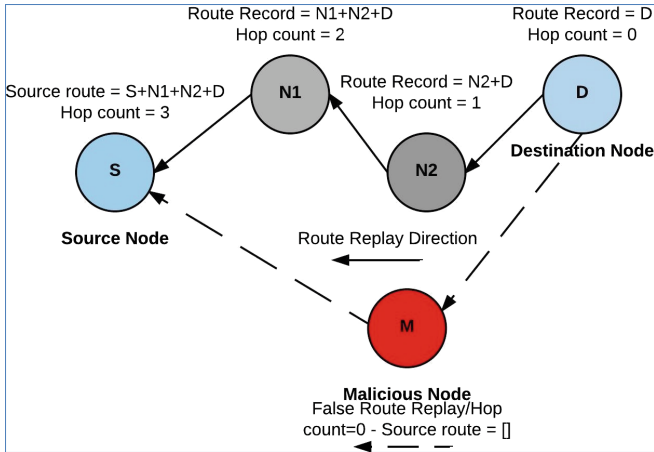
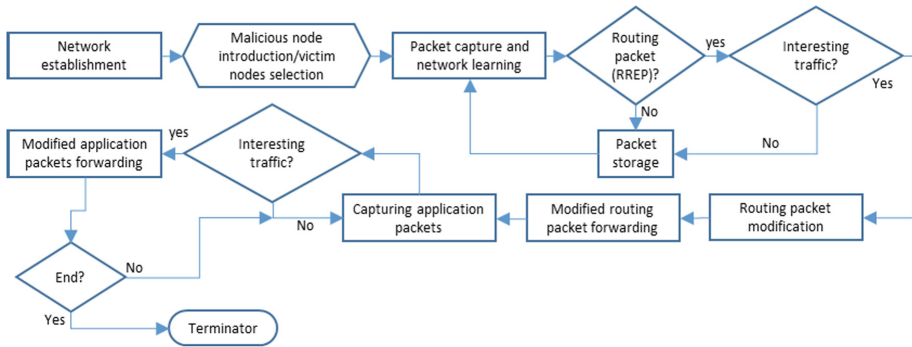


Fig. 3. False route record injection

A wormhole attack begins with an attacker introducing a malicious node into a WSN to gather critical information about network attributes related to node types, network ID, frequency and operational channel. During this step, the target nodes are selected. Subsequently, the malicious node starts a packet capture process to find routing packets involving target nodes (interesting traffic). Once the interesting traffic is captured, the malicious node sets the hop count and relay list to zero in the route record header of the routing packets. Besides, source and destination MAC addresses are changed to match the network addressing of victim nodes since packets from an intermediary node can be captured. Finally, the malicious node forwards the modified routing packet to the destination node, overriding its routing table with the false route and creating a false neighborhood between target nodes in the process. The next step is to continue capturing packets to find application data to be modified and injected into destination nodes. When malicious nodes are not able to capture interesting traffic, the packets are stored and the capture process is restarted. Figure 4 shows the workflow of the proposed algorithm.

In addition, two conditions must be satisfied to carry out a successful wormhole attack: (1) Source and destination addresses must match between layer 2 (802.15.4) and layer 3 (Zigbee); otherwise, the destination node of the RREP discards the packet. (2) The packet sequence number has to be different from the original routing packet; otherwise, the modified packet is discarded [13]. The proposed attack works by overriding the destination node of the RREP’s routing cache by injecting a modified version of the original routing packet, which prevents ZigBee devices from using the original route.



**Fig. 4.** Proposed wormhole attack algorithm.

## 2.1 Attacking Software Design

The proposed algorithm was used to develop an attacking software for real devices as a tool to probe security levels in wireless sensor networks since most research studies describe wormhole attacks by means of simulation environments. On the other hand, the purpose of the attacking software is to expose attributes of ZigBee devices and wormhole attacks that could be used to effectively detect the latter. This section presents a short description of every phase of the attack.

## 2.2 Software Requirements

Scapy and KillerBee frameworks are required to dissect, capture and store packets, and also to inject malicious traffic into victim nodes. These features are combined in a Python script to execute the wormhole attack and build the malicious tunnel.

- (1) Malicious node introduction: During this phase, an attacker sends “beacon-frame”<sup>1</sup> requests channel by channel to discover routing and coordinator nodes in the network, as well as device addressing and network ID using the `zbstumbler` command of the KillerBee framework.
- (2) Attacking software design: The attacking software presents the following attributes and functions.

**Packet Capture and Network Learning:** It occurs when the attacker has selected victim nodes in the network. Then, using relevant networking data like PANID, frequency channel, and node addressing, it captures the packets transmitted over the air through a malicious node. The following pseudocode algorithm describes the packet capture phase.

<sup>1</sup> A “beacon-frame” is a message sent by the coordinator node to synchronize the clocks with network nodes.

**Algorithm 1** Packet capture algorithm

---

**Require:**  $victim\_addr \leftarrow target\_node\_addressing$   
**Require:**  $channel \leftarrow panid\_operational\_channel$   
**Require:**  $filter \leftarrow routing\_header$

```

1: function main ( )
2:   while true do
3:      $pkt \leftarrow KillerBee\_sniffer (channel, filter)$ 
4:     if  $pkt$  is source_routing_packet then
5:       if compare ( $pkt$ ,  $victim\_addr$ ) then
6:          $new\_pkt \leftarrow pkt\_mod (pkt, victim\_addr)$ 
7:          $pkt\_injection (new\_pkt)$ 
8:       else
9:         continue
10:    else
11:      continue

```

The attack begins by using the sniffer object of the KillerBee framework to capture packets with ZigBee source routing header. Once a source routing packet has been captured, the next step determines if the packet belongs to a target device. If not, the while loop continues until KillerBee's sniffer captures a source routing packet that involves victim nodes.

**Interesting Traffic:** A packet is interesting traffic when it is originated or sent from/to an attacker-defined victim device. In that sense, the attacker must dissect the captured packet, extract the addressing data and compare it with victim nodes' addressing. As shown in Algorithm 1, the `compare` function compares addresses. Since the KillerBee sniffer generates an object from the captured packet, packet dissection becomes a simple task. It consists of retrieving the addressing data from the packet object attributes (Algorithm 2).

**Algorithm 2** Interesting traffic algorithm

---

**Ensure:**  $coincidence$

```

1: function compare ( $pkt$ ,  $victim\_addr$ )
2:    $src\_addr \leftarrow pkt.source\_address$ 
3:    $dst\_addr \leftarrow pkt.destination\_address$ 
4:   if  $src\_addr = victim\_addr[0]$  then
5:      $src\_eval \leftarrow 1$ 
6:   else
7:      $src\_eval \leftarrow 0$ 
8:   if  $dst\_addr = victim\_addr[1]$  then
9:      $dst\_eval \leftarrow 1$ 
10:  else
11:     $dst\_eval \leftarrow 0$ 
12:   $coincidence \leftarrow src\_eval \wedge dst\_eval$ 
13:  return  $coincidence$ 

```

---

The previous algorithm determines if a captured packet involves victim nodes' addressing in a data transaction. Once the addressing data is compared, the result can be true if both destination and source addresses of the captured packet and victim nodes are the same. It can also be false if one or more addressing data are not equal. In that case, the packet capture algorithm is executed again.

**Routing Packet Modification:** After finding an RREP packet with the right addressing, the attacking software changes some attributes of the routing information in the captured packet to build the malicious tunnel. It specifically modifies route record information related to hop count, relay list, and sequence number. Algorithm 3 executes the routing packet modification.

**Algorithm 3** Routing packet modification

**Ensure:** new packet

```

1: function pkt_mod(pkt, victim_addr)
2: new_packet ← pkt
3: new_pkt.sequence number ← Random(1, 255)
4: if new_pkt.hop count ≥ 1 then
5:   new_pkt.hop count ← 0
6:   new_pkt.relay list ← []
7:   new_pkt.src address = victim_addr[0]
8:   new_pkt.dst address = victim_addr[1]
9: else if pkt.hop count = 0 then
10:  new_pkt.hop count ← 1
11:  new_pkt.relay list ← [abcd]
12: return new_pkt

```

Packet modification begins by rewriting the *sequence\_number* of the captured packet with a random number between 1 and 255 to prevent the destination node from discarding the modified packet sent by the malicious node. At that point, the wormhole attack can present two scenarios: (1) victim nodes are further apart than one hop of distance, or (2) the victim nodes are neighbors.

The first case describes a classic wormhole attack, and the modifications of *hop\_count* and *relay\_list* are made to “eliminate” the distance between victim nodes. Such changes also make nodes “think” they are neighbors because of the wormhole tunnel. Because victim nodes are distant from each other, layer 2 addressing must be altered to match layer 3 addressing. The second scenario is a “reverse” wormhole attack, where victim nodes are neighbors and a malicious node tries to add distance in-between. In such case, packet modifications are performed by increasing the *hop\_count* number and adding intermediary nodes to the *relay\_list*.

**Routing Packet Forwarding:** After the routing packet has been modified, the next step is to send it to its real destination with the send method of the KillerBee framework. Additionally, a new packet capture process is conducted to search for application data. The latter is used to make further modifications that may cause an unwanted behavior in the application of the WSN. Algorithm 4 shows the packet injection process.



**Algorithm 4** Packet injection algorithm

---

**Require:**  $filter \leftarrow application\_packet\_header$   
**Require:**  $channel \leftarrow panid\_operational\_channel$   
**Require:**  $new\_data \leftarrow attacker\_defined\_data$

```

1: function pkt_injection( $new\_pkt$ )
2:   killerbee_send( $new\_pkt, channel, count \leftarrow 1$ )
3:   while true do
4:      $pkt \leftarrow killerbee\_sniffer(channel, filter)$ 
5:     if compare( $pkt, victim\_addr$ ) then
6:        $pkt.data = new\_data$ 
7:       killerbee_send( $pkt, channel, count \leftarrow 1$ )
8:     else
9:       continue

```

---

Packet injection causes two possible effects in victim nodes because, once the modified packet is processed by the destination node, it depends on the malicious node whether to forward the next application packets or not. If they are not forwarded, the attack may cause a denial-of-service (DoS) state.

**Data Packets Modification and Forwarding:** As shown in Algorithm 4, this wormhole attack tries to modify application as well as routing data. In this case, the destination node of the application data would receive the attacker's data. The main differences with a replication attack is that the proposed wormhole prevents the direct communication between involved victim nodes and it works over real-time traffic.

At last, the entire process is repeated indefinitely, injecting false routes with every modified data packet sent to the destination node to maintain the wormhole tunnel until the script is stopped or moved to another network point.

### 3 Implementation and Results

In this section, the implementation of the proposed worm-hole attack on a testing network takes place without encryption protocols applied in the packets to measure its impact on unsecured devices.

#### 3.1 Network Requirements and Characteristics

Table 1 lists legitimate features of nodes and the parameters used to build the prototype network. The malicious node specifications are shown in Table 2. Atmel RZUSB STICK with KillerBee firmware is used in conjunction with Raspberry Pi 3 to capture packets and inject modified data and routing packets into victim nodes.

In order to execute the reverse and classic wormhole attacks, two testing networks were built with a coordinator node and two router nodes. Figure 5 presents a reverse wormhole scenario with router nodes sharing a direct link, which is common between

**Table 1.** Legitimate node features.

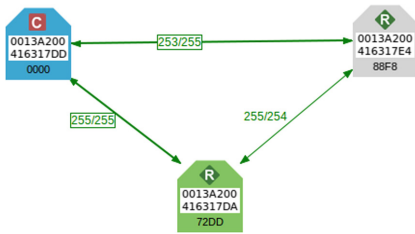
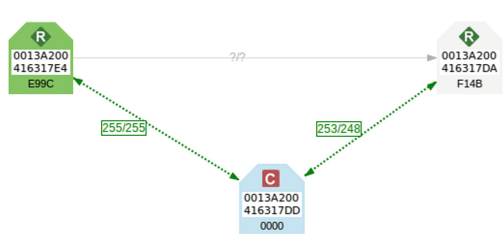
Type	XBee S2C (XB24C)
Firmware	405E
Functions set	ZIGBEE TH Reg
Medium access control	IEEE 802.15.4
Network layer	ZigBee (Source Routing)
Frequency	2.4 GHz
Router nodes	2
Coordinator nodes	1
Network ID (PANID)	10
Microcontroller	Arduino UNO - ATMEGA 328p

**Table 2.** Malicious node features.

Node type	Raspberry Pi 3 Model B
Network interface	ATAVRRZUSBSTICK
Firmware	Killerbee
Scripting language	Python 2.7.14
Frameworks	Scapy - Killerbee
Operating system	Raspbian

neighboring nodes. On the other hand, Fig. 6 shows router nodes without a direct link and the coordinator node as an intermediary node (adding one hop of distance between router nodes) to test a classic wormhole attack.

In a reverse wormhole attack, victim nodes are identified by network addresses 0x72DD (source of route record) and 0x88F8 (source of application data). In a classic wormhole attack, the source node has the network address 0xE99C, while the destination node has 0xF14B. At last, the coordinator node has the default address 0x0000 in both cases.

**Fig. 5.** Prototype network for reverse wormhole attack**Fig. 6.** Prototype network for classic wormhole attack.

### 3.2 Wormhole Attack Execution

- (1) *Reverse wormhole attack:* The main goal is to add distance between victim nodes by modifying the hop count and relay list in the routing packet, thus avoiding

using the direct link shared by nodes 0x72DD and 0x88F8. The following command line output shows the execution of the wormhole attack script.

```
$ sudo python wormhole.py 0x72DD 0x88F8 14-e -f "reverse wormhole"
[**]Enter number of hops:1
[**]Enter comma separated hops:abcd
[**]Sniffing and searching for source routing...
[OK]Route record found for src'0x88f8': [OK]Sequence Number:199
[OK]Route record parameters:
    source_addr:'0x72DD'
    addresses:[]
    hop_count:0
    options:0
    id: route record
[**]Injecting route record to 88f8 Sent 1 packets.
[->)Fake route injected!
[**]Sniffing for application data... Sent 1 packets.
```

The attack starts by capturing packets until a source routing packet involving victim nodes is found. Then, the routing packet is injected a hop count equal to 1 and an intermediary node (0xABCD) into its relay list parameter. Finally, when the script captures an application packet, the application data is replaced with the sentence “reverse wormhole”. Figure 7 shows the original application frame sent by node 0x88F8. In this case, the original application packet has the word “TEST”. When the packet arrives at the destination node, an update of the source route is sent to 0x88F8 from 0x72DD, as shown in Fig. 8.

16-bit dest. address
72 DD
Broadcast radius
00 (0)
Options
00
RF data
ASCII <input checked="" type="checkbox"/> HEX
TEST

Frame type
A1 (Route Record Indicator)
64-bit source address
00 13 A2 00 41 63 17 DA
16-bit source address
72 DD
Receive options
00
Number of addresses
00 (0)

Fig. 7. Original application packet payload.

Fig. 8. Original source routing packet for neighboring nodes.

Figure 9 shows the malicious route injected into 0x88F8 when the reverse wormhole attack captures the first source routing packet.

<b>16-bit source address</b>
72 DD
<b>Receive options</b>
00
<b>Number of addresses</b>
01 (1)
<b>Address 1</b>
AB CD

**Fig. 9.** Modified source routing packet.

<b>Cluster ID</b>
00 11
<b>Profile ID</b>
C1 05
<b>Receive options</b>
01
<b>RF data</b>
ASCII <input type="checkbox"/> HEX <input type="checkbox"/>
reverse wormhole

**Fig. 10.** Malicious data received at destination node.

Figures 8 and 9 show the difference between both routes. The first one contains the attributes of the original route with 0 relays as “Number of addresses”. The second one contains the false intermediary nodes with a relay that has the address 0xABCD. Due to this, the malicious node is the only one that can listen to the next application packets sent by 0x88F8, which are changed by the attacker’s malicious data (Fig. 10).

- (2) *Classic wormhole attack:* Similar to a reverse wormhole attack, this variant captures routing packets to create the malicious tunnel and application packets to inject malicious data. Figure 11 shows the legitimate and malicious routing packet received at the source node. The first route record indicator entry belongs to the original source of RREP and the second entry is the RREP modified by the malicious node to override the routing table of a victim node.

ID	Time	Length	Frame
0	16:28:35.493	18	Transmit Request
1	16:28:35.830	15	Route Record Indicator
2	16:28:35.870	7	Transmit Status
3	16:28:36.091	13	Route Record Indicator

**Fig. 11.** Received routing packets.

Once again, a source route is updated when the source node attempts to send the word “TEST” and the packet is captured and modified by the wormhole attack. Figures 12 and 13 present the changes in the route received by the source node.

<b>16-bit source address</b>
F1 4B
<b>Receive options</b>
00
<b>Number of addresses</b>
01 (1)
<b>Address 1</b>
00 00
<b>Checksum</b>
D7

**Fig. 12.** Original source route fields.

<b>16-bit source address</b>
F1 4B
<b>Receive options</b>
00
<b>Number of addresses</b>
00 (0)

**Fig. 13.** False source route fields.

An evident change can be observed in the field *Number of addresses* (hop count) of the source routes: the value goes from 1 hop in the first packet to 0 hops in the second. After false route injection, the source node attempts to send the word “TEST” and the task of the wormhole attack script is to replace these data with the word “WORMHOLE”. Figure 14 shows the malicious data packet received by the destination node, and Fig. 15 shows the content of the packet.

ID	Time	Length	Frame
← 0	16:28:36.277	42	Explicit RX Indicator
← 1	16:28:37.177	12	Many to One Route Request Indicator

**Fig. 14.** Modified application data received at destination node.

**Profile ID**  
C1 05

**Receive options**  
01

**RF data**

ASCII  HEX

WORMHOLE

**Fig. 15.** Application data content after attack.

### 3.3 Signatures for Wormhole Attack Detection

- (1) *Routing packet duplication:* In ZigBee devices, source routes can be requested by sending the Network Discovery (ND) command or updated when destination nodes receive a packet. In that sense, a wormhole attack must inject false routes for every modified packet that is sent, thus forcing the sensors/devices to receive two source routing packets per data packet transmitted to destination nodes. The abrupt changes in route record fields of the routing packets and the increase in transmitted routing packets could be used to detect the presence of an attacker in the network.
- (2) *Multiple “beacon-frame” requests without a joined de- vice:* The first step to attack WSNs is launching a discovering process to identify possible targets in the network. In 802.15.4/Zigbee networks, “beacon-frame” requests are responded by router and coordinator nodes to have new nodes join the network. However, after malicious nodes send a “beacon-frame” request, no new devices join the network. To monitor this behavior, pairing beacon request frames with newly joined devices in the WSN would help to detect active scans before the wormhole attack occurs.
- (3) *Neighborhood table and link status packets:* ZigBee devices regularly send link status packets to maintain a first hop neighborhood table. Due to the fact that remote nodes cannot share link status packets, wormholes are detected by examining previous link status messages of nodes in a routing packet with route record of zero hops. If previous link status messages are not found, a wormhole threat is detected. On the other hand, a reverse wormhole is detected by checking

routing packets with route records containing more than one hop. If the nodes involved in the transmitted packet have shared link status messages before, a reverse wormhole is detected. This approach could be used with neighborhood tables instead of link status messages.

### 3.4 Recommendations

Due to the harmful behavior of a wormhole attack, the cryptographical features of the ZigBee specification should avoid modifying data and routing packets during wireless transmission. Besides, encryption keys must be regularly changed to prevent brute force attacks and reduce the functionality of possible key extraction from a stolen node. Additionally, a better randomization method for the sequence number in every packet must be implemented by the ZigBee specification to make predicting this number difficult and prevent packet injection attacks, which causes packets with a wrong sequence number to be discarded by legitimate nodes.

## 4 Conclusions and Future Work

This implementation of a wormhole attack in real devices was successful in using the algorithm proposed to manipulate packets with the KillerBee framework and Scapy decoders. Besides, a new variant of the wormhole attack was introduced and tested to show the flexibility and risk of malicious nodes in a network. Such variant takes advantage of the vulnerability of ZigBee devices for wormhole attacks and packet injection. On the other hand, the lack of effective security measures for WSNs must be explored from an empirical point of view to close the security gap of IoT with the available technology. This would also enable end users to implement security tools for real devices. As future work, an Intrusion Detection System (IDS) for wormhole attacks is going to be designed and implemented using signatures and patterns presented in this paper as result of the wormhole attack execution on real devices. Additionally, other experimental attacks, such as sinkhole and Sybil, will be explored to improve the detection system.

## References

1. Sahmim, S., Gharsellaoui, H.: Privacy and security in internet-based computing: cloud computing, internet of things, cloud of things: a review. *Procedia Comput. Sci.* **112**, 1516–1522 (2017)
2. Rani, A., Kumar, S.: A survey of security in wireless sensor networks, pp. 3–7 (2017)
3. Zhu, C., Leung, V.C.M., Shu, L.E.I.: Green internet of things for smart world. *IEEE Access* **3**, 2151–2162 (2015)
4. Patle, A.: Vulnerabilities, attack effect and different security scheme in WSN: a survey (2016)
5. Goyal, S.: Wormhole and sybil attack in WSN: a review, pp. 1463–1468 (2015)

6. Anwar, R., Bakhtiari, M., Zainal, A., Abdullah, A.H., Naseer Qureshi, K.: Security issues and attacks in wireless sensor network. *World Appl. Sci. J.* **30**(10), 1224–1227 (2014). ISSN 1818-4952
7. Jao, M., et al.: A wormhole attacks detection using a QTS algorithm with MA in WSN (2015)
8. Hu, Y., Perrig, A., Johnson, D.B.: Wormhole attacks in wireless Networks. *IEEE J. Sel. Areas Commun.* **24**(2), 370–380 (2006)
9. Jabeur, N., Sahli, N., Muhammad, I.: survey on sensor holes: a cause-effect-solution perspective. *Procedia - Procedia Comput. Sci.* **19**, 1074–1080 (2013)
10. Amish, P., Vaghela, V.B.: Detection and prevention of wormhole attack in wireless sensor network using AOMDV protocol. *Procedia - Procedia Comput. Sci.* **79**, 700–707 (2016)
11. Qazi, S., Raad, R., Mu, Y., Susilo, W.: Securing DSR against wormhole attacks in multirate ad hoc networks. *J. Netw. Comput. Appl.* **36**, 582–592 (2013)
12. Bhagat, S.: A detection and prevention of wormhole attack in homogeneous wireless sensor network, pp. 1–6 (2016)
13. Patel, A., Patel, N., Patel, R.: Defending against wormhole attack in MANET. In: 2015 Fifth International Conference on Communication Systems and Network Technologies (CSNT), pp. 674–678 (2015)
14. Zheng, J., Qian, H., Wang, L.: Defense technology of wormhole attacks based on node connectivity. In: 2015 IEEE International Conference on Smart City/SocialCom/SustainCom, pp. 421–425 (2015)
15. ZigBee Alliance: ZIGBEE Specification. ZigBee Alliance Board of Directors (ZigBee Standards Organization). Document 053474r17 (2008)
16. Johnson, D.B.: The Dynamic Source Routing Protocol for Mobile Ad Hoc Networks (DSR) (2004)