Zigator: Analyzing the Security of Zigbee-Enabled Smart Homes

Dimitrios-Georgios Akestoridis
Carnegie Mellon University
Moffett Field, California, USA
akestoridis@cmu.edu

Madhumitha Harishankar
Carnegie Mellon University
Moffett Field, California, USA
mharisha@cmu.edu

Michael Weber
Carnegie Mellon University
Moffett Field, California, USA
mikex@cmu.edu

Patrick Tague
Carnegie Mellon University
Moffett Field, California, USA
tague@cmu.edu

ABSTRACT

As the popularity of Internet-connected devices for residential use increases, it is important to ensure that they meet appropriate security goals, given that they interact with the physical world through sensors and actuators. Zigbee is a wireless communication protocol that is commonly used in smart home environments, which builds on top of the IEEE 802.15.4 standard. In this work we present a security analysis tool, called Zigator, that enables in-depth study of Zigbee networks. In particular, we study the security consequences of the design choice to disable MAC-layer security in centralized Zigbee networks. We show that valuable information can be gained from passive inspection of Zigbee traffic, including the identification of certain encrypted NWK commands, which we then use to develop selective jamming and spoofing attacks. An attacker may launch these attacks in order to force the end user to factory reset targeted devices and eventually expose the network key. We validated our attacks by setting up a testbed, using open-source tools, that incorporates commercial Zigbee devices. Finally, we publicly release the software tools that we developed and the Zigbee packets that we captured, to contribute back to the research community.

CCS CONCEPTS

• **Security and privacy → Mobile and wireless security**; • **Networks → Mobile and wireless security**; **Home networks**; *Mobile ad hoc networks*; *Sensor networks*.

KEYWORDS

Zigbee, IEEE 802.15.4, smart home, security analysis, jamming

ACM Reference Format:
Dimitrios-Georgios Akestoridis, Madhumitha Harishankar, Michael Weber, and Patrick Tague. 2020. Zigator: Analyzing the Security of Zigbee-Enabled Smart Homes. In *13th ACM Conference on Security and Privacy in Wireless and Mobile Networks (WiSec '20), July 8–10, 2020, Linz (Virtual Event), Austria.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3395351.3399363

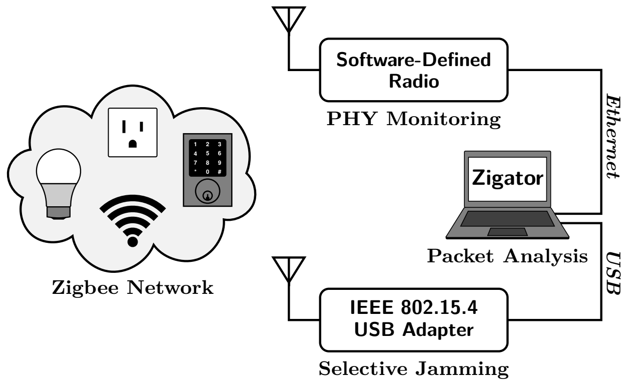


Figure 1: Zigator analyzes captured Zigbee packets to enable the development of selective jamming and spoofing attacks with an IEEE 802.15.4 USB adapter, which we study by capturing I/Q signals with a software-defined radio.

1 INTRODUCTION

Smart homes are an application of the Internet of Things (IoT) [3] where everyday devices are connected to the Internet as either sensors, actuators, or both. This level of connectivity enables residents to monitor the state of their devices, issue commands to change their states, as well as automate common tasks. Several communication protocols may operate concurrently in a smart home to facilitate this, mainly because different devices have different requirements, that are usually bridged by a smart hub. Some devices require a high-data-rate connection to operate satisfactorily, while others have relaxed throughput requirements so the focus is shifted to low power consumption and low manufacturing cost. But along with the benefits that smart home devices bring, they also raise serious security concerns because they interact with the physical world. Breaching the security of their communication protocols can thus impact the physical security of the residents.

One of the most widely used communication protocols for IoT devices, especially in smart home environments, is Zigbee [43]. The Zigbee protocol defines the upper layers of the IEEE 802.15.4 standard to provide low-data-rate wireless connectivity to low-cost devices with low power consumption. The IEEE 802.15.4 standard defines the PHY and MAC layers of the protocol stack, with home automation being one of its main applications [9]. The operation of

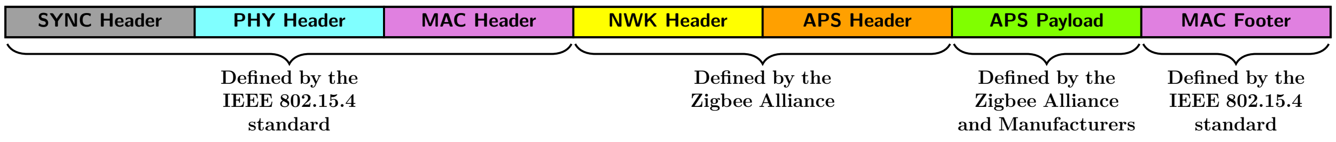

SYNC Header
PHY Header
MAC Header
NWK Header
APS Header
APS Payload
MAC Footer
Defined by the IEEE 802.15.4 standard
Defined by the Zigbee Alliance
Defined by the Zigbee Alliance and Manufacturers
Defined by the IEEE 802.15.4 standard

Figure 2: High-level view of a general Zigbee packet without any Auxiliary Header or any Message Integrity Code.

Zigbee networks is defined by the Zigbee Alliance, a group of companies that also certifies Zigbee products which are supported by numerous smart home ecosystems including Samsung SmartThings and Amazon's Echo Plus [42]. However, the Zigbee Alliance does not impose security decisions on manufacturers [45, p. 442]. Furthermore, Zigbee networks remain largely unmonitored in smart homes. Hence, the development of comprehensive security analysis tools is of paramount importance to independent researchers.

Several attacks have been demonstrated against Zigbee networks including device hijacking [23], a Zigbee worm [27], and, more recently, remote code execution [10]. The aforementioned demonstrations are, for the most part, limited to distributed Zigbee networks because they exploit vulnerabilities specific to the distributed security model such as (a) the use of unsecured factory-reset commands, (b) knowledge of a leaked encryption key under non-disclosure agreement, and (c) the use of proximity-based commissioning. These are inherently absent in centralized Zigbee networks, where a single device takes the role of the Trust Center that manages encryption keys and authorizes devices requesting to join the network. Even though centralized Zigbee networks are recommended for higher security [48], they have received less attention from the research community, especially since Zigbee 3.0 devices have been commercially available that address some prior vulnerabilities. To add to the challenge, there is a lack of robust security analysis tools for Zigbee networks, commercial Zigbee devices use closed-source software, and certain specification documents have not been released to the public. Previous studies that incorporate centralized Zigbee networks largely demonstrate information leakage from encrypted traffic [1, 17, 35, 41], such as the identification of triggered events, without exploring how an attacker may use this information to disrupt the operation of centralized Zigbee networks. While several command injection attacks have been demonstrated against centralized Zigbee networks [8, 21, 49], these attacks require knowledge of the network key. However, to the best of our knowledge, realistic methods by which an attacker can acquire this encryption key have not been thoroughly explored.

In this work we present a security analysis tool, called Zigator, that we developed in order to study the exposure of Zigbee networks to passive and active attacks due to the design choice to not utilize security services on the MAC layer. In fact, the Zigbee PRO 2015 specification states that NWK commands should disable MAC-layer security "since any secured frame originating from the NWK layer shall use NWK layer security" [45, p. 269]. We show that an attacker can infer valuable information from operational Zigbee networks, including the logical device type of every Zigbee device. In addition, despite the use of encryption on the NWK layer, we show that an attacker can identify half of all possible NWK commands with 100% accuracy. We use this information to develop selective jamming and spoofing attacks that an attacker can launch in order to eventually gain access to the network key. Furthermore, we recommend security enhancements for the commissioning process of Zigbee 3.0 networks to protect against this attack vector. We validated our attacks by building a testbed, illustrated in Fig. 1, that incorporates commercial Zigbee devices. We contribute to the research community by publicly releasing Zigator, our dataset, and our modifications to several open-source tools[1]. Note that even though this work mainly focuses on centralized Zigbee networks, some of our contributions can also be applied to distributed Zigbee networks (e.g., the selective jammer described in Section 4.3).

The rest of this paper is organized as follows. We provide relevant background information in Section 2 and delineate our threat model and assumptions in Section 3. Section 4 details the instrumentation of our testbed, while Section 5 presents an overview of Zigator. We describe our experimental setup in Section 6 and report the results of the passive and active attacks that we launched in Section 7. Finally, we review related work in Section 8 and conclude in Section 9.

2 BACKGROUND

In this section we provide a brief overview of the packet format, security models, and logical device types in Zigbee networks.

Packet format. The operation of Zigbee networks is defined over multiple documents [18, 45–47], not all of which are publicly available[2]. The general format of a Zigbee packet, without any security features, is shown in Fig. 2. The actual format of Zigbee packets highly varies because there are multiple packet types that are identified using header fields from the Medium Access Control (MAC), Network (NWK), and Application Support (APS) layers. These packet types include MAC acknowledgments, MAC beacons, MAC commands, NWK commands, APS acknowledgments, and APS commands. Furthermore, the payload of the APS layer may be a Zigbee Device Profile (ZDP) command or a, potentially proprietary, Zigbee Cluster Library (ZCL) command. Additional identification fields are used to further differentiate these commands and determine the format of their payloads. Zigbee provides security services for packets on its NWK and APS layers by including an auxiliary header after the header of the corresponding layer and a message integrity code after its payload. Zigbee uses the CCM* block cipher mode [45, p. 456] to encrypt the payload and authenticate the header and payload of the corresponding layer using the AES-128 algorithm [24], according to the security level[3] of the security control field [45, p. 425]. The CCM* mode extends the CCM mode [12] by adding an encryption-only security level. However, the Zigbee

[1]Additional resources are available at http://mews.sv.cmu.edu/research/zigator/.
[2]For example, the Zigbee PRO 2017 specification has not been released to the public.
[3]Zigbee devices overwrite the security level with zeros before transmission, so the receiver has to restore it for the decryption and verification process [45, pp. 381, 387].

Alliance advises against the use of this security level [45, p. 425] since it is vulnerable to a single-packet denial-of-service attack [30]. Thus, an implementation of AES-128 in CCM mode can decrypt and verify typical Zigbee traffic. The nonce consists of the source's 64-bit IEEE address, a 32-bit frame counter, and the 8-bit security control field [45, p. 427]. The encryption key that is shared across all devices in a Zigbee network is referred to as the network key, while the encryption keys that are used only between pairs of devices are referred to as link keys. The IEEE addresses of Zigbee devices are referred to as their extended addresses, while their locally assigned addresses are referred to as their short addresses. Finally, each Zigbee network uses a Personal Area Network Identifier (PAN ID), which should be unique within its coverage area [45, p. 233].

Security models. In order to provide a balance between security and usability, Zigbee supports two security models: distributed and centralized. Distributed Zigbee networks aim for ease of use and consist of Zigbee Routers and Zigbee End Devices, with each Zigbee Router being able to issue encryption keys. Centralized Zigbee networks, which are recommended for higher security and are the main focus of this work, include a Zigbee Coordinator that typically takes the role of the Trust Center in order to manage encryption keys and authorize devices that request to join the network. Depending on the security model, different types of link keys can protect the transportation of the network key [46, p. 30].

Logical device types. Zigbee Coordinators are typically mains-powered devices that can form centralized networks but cannot join other networks. Zigbee Routers are typically mains-powered devices that can form distributed networks but can also join other networks. Both of these logical device types can route packets for other Zigbee devices. Zigbee End Devices are usually battery-powered devices that cannot form either centralized or distributed networks, but can join other networks. Zigbee End Devices usually keep their receivers disabled when they are idle to conserve energy and rely on the Zigbee Coordinator or a Zigbee Router for routing services. A typical hardware device that operates as a Zigbee Coordinator is the smart hub, which may provide access to the Internet and other local networks. Examples of hardware devices that operate as Zigbee Routers include power outlets and light bulbs, while door locks and motion sensors may operate as Zigbee End Devices.

3 THREAT MODEL AND ASSUMPTIONS

In keeping with standard security literature, we set the following security objectives for Zigbee networks:

- **Authenticity.** Unauthorized devices should not be able to impersonate authorized devices.
- **Integrity.** The devices should be able to detect and reject tampered packets.
- **Confidentiality.** Unauthorized devices should not have access to sensitive information.
- **Availability.** Networking services should be available to the devices when they are needed.

We assume that the end user and their devices are trusted in the sense that they do not deliberately downgrade the security of the Zigbee network, with the physical security of the devices being outside the scope of this work. The attacker is an outsider that may utilize more powerful hardware than that of the network's devices,

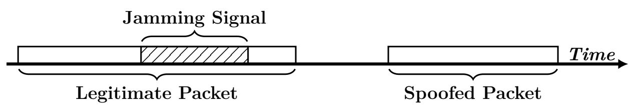


Figure 3: An attacker may interfere with the transmission of a legitimate packet and then inject a spoofed packet.

such as high-gain directional antennas and a laptop computer. We assume that the attacker has no prior knowledge of any network key, but they have knowledge of the default Trust Center link key since it is publicly available [45, p. 377]. Additionally, we consider the possibility of the attacker having access to a subset of install codes, as we discuss in Section 7.3. Furthermore, we assume that the attacker is not present during the initial formation of the Zigbee network. If that were the case, violation of the aforementioned security objectives is trivial as long as a single legacy Zigbee device exists in the network or the install code of a Zigbee 3.0 device is known. Finally, our discussions in the following sections assume that the end user's Zigbee network is using the centralized security model. Note that we do not take into account Zigbee End Devices that keep their receivers enabled when they are idle, low-power routers, Green Power devices, or beacon-enabled networks. Although these configurations are supported according to the Zigbee PRO 2015 specification [45], to the best of our knowledge, they are not widely used in smart home environments at the time of writing.

Ultimately, the goal of the attacker is to obtain the network key from an already formed Zigbee network. Currently, the security of Zigbee-enabled smart homes depends heavily on its secrecy. An attacker that gains access to the network key can decrypt most of the encrypted payloads and inject commands that change the state of the end user's devices, violating our security objectives.

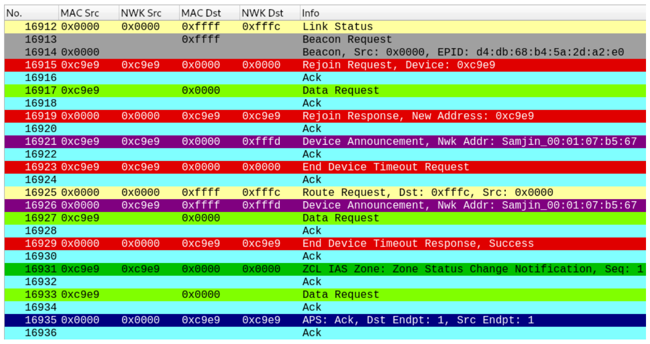
4 TESTBED INSTRUMENTATION

There are three core functionalities that we instrument in our testbed to analyze the security of operational Zigbee networks: (a) packet sniffing, (b) packet injection, and (c) packet jamming. These form the building blocks for the development of more sophisticated attacks. For example, as illustrated in Fig. 3, an attacker can monitor Zigbee traffic until a packet of interest is being transmitted in order to interfere with its transmission and then inject a spoofed packet. We demonstrate such an attack at the end of this section.

4.1 Packet Sniffing

The most fundamental functionality that we require is the ability to capture Zigbee packets. We used a USRP N210 [14] to capture I/Q signals and perform demodulation in software. In particular, we used GNU Radio [15] with the `gr-ieee802-15-4` module [5] to receive IEEE 802.15.4 packets and then store them in PCAP format with the `gr-foo` module [4]. The main reason why we opted to use a software-defined radio, instead of an IEEE 802.15.4 USB adapter, is that it also allows us to analyze the effectiveness of packet jamming.

The resulting PCAP files were inspected using Wireshark [37], which can decrypt encrypted payloads if the corresponding encryption keys are provided. However, there is no widespread use of a suitable configuration profile. Certain header fields are vital for the



No.	MAC Src	NWK Src	MAC Dst	NWK Dst	Info
16912	0x0000	0x0000	0xffff	0xfffc	Link Status
16913			0xffff		Beacon Request
16914	0x0000				Beacon, Src: 0x0000, EPID: d4:db:68:b4:5a:2d:a2:e0
16915	0xc9e9	0xc9e9	0x0000	0x0000	Rejoin Request, Device: 0xc9e9
16916					Ack
16917	0xc9e9		0x0000		Data Request
16918					Ack
16919	0x0000	0x0000	0xc9e9	0xc9e9	Rejoin Response, New Address: 0xc9e9
16920					Ack
16921	0xc9e9	0xc9e9	0x0000	0xfffd	Device Announcement, Nwk Addr: Samjin_00:01:07:b5:67
16922					Ack
16923	0xc9e9	0xc9e9	0x0000	0x0000	End Device Timeout Request
16924					Ack
16925	0x0000	0x0000	0xffff	0xfffc	Route Request, Dst: 0xfffc, Src: 0x0000
16926	0x0000	0xc9e9	0xffff	0xfffd	Device Announcement, Nwk Addr: Samjin_00:01:07:b5:67
16927	0xc9e9		0x0000		Data Request
16928					Ack
16929	0x0000	0x0000	0xc9e9	0xc9e9	End Device Timeout Response, Success
16930					Ack
16931	0xc9e9	0xc9e9	0x0000	0x0000	ZCL IAS Zone: Zone Status Change Notification, Seq: 1
16932					Ack
16933	0xc9e9		0x0000		Data Request
16934					Ack
16935	0x0000	0x0000	0xc9e9	0xc9e9	APS: Ack, Dst Endpt: 1, Src Endpt: 1
16936					Ack

Figure 4: Captured packets during the rejoin process of a Zigbee 3.0 device, colored according to the rules of our Wireshark configuration profile. The text was slightly enlarged and several columns were hidden due to space constraints.

understanding of communication flows, while coloring rules ease the inspection process through the emerged patterns. To that end, we developed a Wireshark configuration profile that is tailored for Zigbee traffic[4]. For the coloring rules, we decided to group certain packet types. For example, we grouped commands that affect the membership and encryption keys of the devices (e.g., Association Request, Rejoin Request, and Transport Key commands), while Data Requests were colored uniquely because, as we describe in Section 7.1, they provide valuable information about the logical device types of their transmitters and receivers. Fig. 4 showcases a few of these coloring rules in Wireshark's packet list pane.

4.2 Packet Injection

The next required functionality is the injection of arbitrary packets. Several header fields that are used in Zigbee packets have been implemented in Scapy [33], which enables us to easily forge arbitrary packets. Fig. 5 depicts a simplified GNU Radio Companion flow graph that enables the reception and logging of IEEE 802.15.4 packets with one antenna and the transmission of forged IEEE 802.15.4 packets that are sent over a UDP connection with another antenna. We release a collection of GNU Radio Companion flow graphs[5] that we used in order to conduct the experiments of this paper.

However, this approach is not suitable for the transmission of time-critical packets due to the delay introduced by transferring packets to the host machine and then back to the transceiver [31]. As a result, we modified the firmware of an ATUSB [25] in order to transition from the receive state to the transmit state and send the forged packet within the required time frame. For instance, a Zigbee device that uses the O-QPSK PHY layer of the IEEE 802.15.4-2011 standard in the 2.4 GHz band will wait 864 microseconds to receive a requested MAC acknowledgment [18, p. 129].

4.3 Packet Jamming

The third functionality that we need is the ability to interfere with the transmission of legitimate packets. We reproduced Bloessl's implementation of a selective jammer [6], which selectively jams

[4]Available at https://github.com/akestoridis/wireshark-zigbee-profile.
[5]Available at https://github.com/akestoridis/grc-ieee802154.

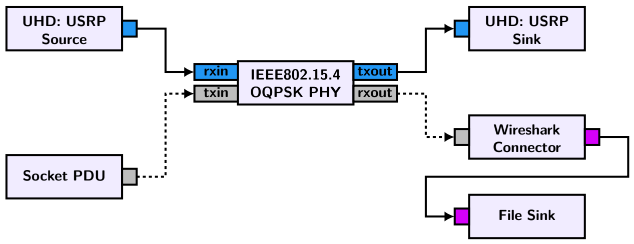


Figure 5: A simplified GNU Radio Companion flow graph that uses an implementation of the IEEE 802.15.4 O-QPSK PHY layer [7] to transmit and receive Zigbee packets.

all packets that are destined for a specific device. However, this implementation does not provide sufficient control over the jamming condition to launch the attacks that we present in Section 7.3. To this end, we modified the ATUSB firmware [26] as follows:

- Whenever an RX_START interrupt is detected, which is issued typically 9 microseconds after a PHY header has been received [2, p. 158], we retrieve the length of the packet that we are receiving from the frame buffer.
- Then we wait for 32 microseconds, i.e. the time that it takes for the transmission of a single byte [18, p. 163], and after that we retrieve the next byte from the frame buffer[6].
- We repeat the previous process until we can determine whether the jamming condition is satisfied or not.
- If the jamming condition is satisfied, we force a transition from the BUSY_RX state to the PLL_ON state, using the FORCE_PLL_ON command, and then we transmit an arbitrary packet by transitioning to the BUSY_TX state, whose length depends on the length of the receiving packet and the number of bytes that we have processed.
- After that, we transition back to the RX_ON state and wait for the next RX_START interrupt.

We also provide a visual description of our jammer's behavior in Fig. 6. Note that our implementation[7] supports the definition of arbitrary jamming conditions for the selective jammer, which enabled us to launch our proof-of-concept attacks.

We provide Fig. 7, which shows the magnitude of a captured I/Q signal, to demonstrate how selective jamming can be combined with packet injection to launch more sophisticated attacks. The jamming condition of this experiment was satisfied only for packets of a specified network that request a MAC acknowledgment. In order to be able to spoof a MAC acknowledgment after interfering with the transmission of a legitimate packet, our jammer stored its MAC sequence number when it was received. The jamming packet was then transmitted, while the spoofed MAC acknowledgment was transmitted shortly after the transmission of the legitimate packet was completed. It is important to note that even though Zigbee devices use secured APS acknowledgments to protect user commands, MAC and NWK commands are still relying on unsecured MAC acknowledgments. We take advantage of this fact to develop the attacks that we present in Section 7.3.

[6]This type of read access is supported by the AT86RF231 transceiver [2, p. 126].
[7]Available at https://github.com/akestoridis/atusb-attacks.

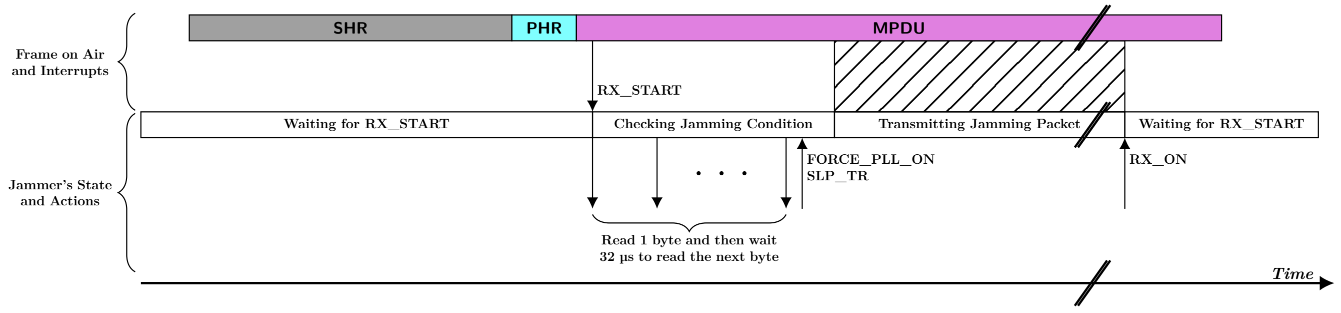

SHR
PHR
MPDU
Frame on Air and Interrupts
RX_START
Waiting for RX_START
Checking Jamming Condition
Transmitting Jamming Packet
Waiting for RX_START
FORCE_PLL_ON
SLP_TR
RX_ON
Jammer's State and Actions
Read 1 byte and then wait 32 μs to read the next byte
Time

Figure 6: Our selective jammer provides fine-grained control over the jamming condition by reading the bytes of a transmitted packet as they are being received to determine whether it will be jammed or not before the completion of its transmission.

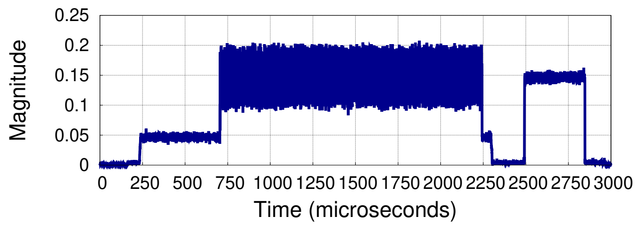


Figure 7: Captured I/Q signal during the selective jamming of a Zigbee packet and injection of a MAC acknowledgment.

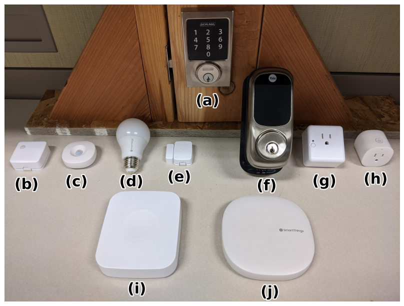


Figure 8: The commercial Zigbee devices that we studied.

5 SECURITY ANALYSIS WITH ZIGATOR

In this section we provide an overview of Zigator, the software tool that we developed in Python to analyze the security of Zigbee networks[8]. Zigator's main dependency is the Scapy library [33]. Specifically, Zigator uses the `dot15d4` and `zigbee` layers of the Scapy library to parse captured Zigbee packets and forge new ones for injection. However, at the time of writing, several header fields and frame types have not been implemented in the latest version of Scapy (v2.4.3) that are vital to our research. Hence, we forked Scapy and implemented these enhancements for Zigator[9].

PyCryptodome [13] is another library that Zigator heavily uses. After parsing the unencrypted header fields of an encrypted packet, Zigator constructs the nonce and authenticated section of the packet in order to decrypt and verify it using AES-128 in CCM mode. We also used PyCryptodome's implementation of AES-128 to implement the Matyas-Meyer-Oseas (MMO) hash function [22]. Since Zigbee uses message digests of the same length as the block size of AES-128 [45, p. 460], the message digest H_t of a bitstring x, that is padded and divided into t 128-bit blocks, is computed iteratively as

$$H_i = E_{H_{i-1}}(x_i) \oplus x_i,\ 1 \le i \le t, \tag{1}$$

where $H_0 = 0$ and $E_K(\cdot)$ denotes the encryption function of AES parameterized by a 128-bit key K. The MMO hash function is used in Zigbee 3.0 networks to derive the preconfigured Trust Center link key of Zigbee 3.0 devices from their install codes [46, p. 75]. Furthermore, the MMO hash function is also used to define Zigbee's keyed-hash message authentication code (HMAC) [45, p. 461] to derive the encryption key of certain APS-layer protected packets, which has also been implemented in Zigator.

Zigator stores almost all possible header fields of the captured packets in an SQLite database, along with other metadata and inferred information that we discuss in Section 7.1. This approach enables us to gain valuable insight into the nature of Zigbee traffic because it allows us to execute detailed SQL queries. Zigator also provides a feature-rich command-line interface, which allows the security analyst to visualize their data as well as send forged packets for injection over UDP. In addition, Zigator can train decision tree classifiers, using the Scikit-learn library [32], to distinguish different packet types, which provided vital information for the development of the generalized decision tree that we present in Section 7.2. Subsequently, we present our experimental setup and analyze the security of Zigbee networks using Zigator.

6 EXPERIMENTAL SETUP

We captured packets that were generated from ten commercial Zigbee devices, shown in Fig. 8. We used the third generation of the SmartThings Hub (IM6001-V3P01) as our Zigbee Coordinator for half of our experiments, while the second generation of the SmartThings Hub (STH-ETH-200) was used for the other half (see Fig. 8j and 8i respectively). It should be noted that only the former can commission Zigbee 3.0 devices using their install codes; the latter uses the default Trust Center link key to commission them. The Zigbee 3.0 devices that we used are a SmartThings Outlet (IM6001-OTP01), a SmartThings Motion Sensor (IM6001-MTP01), a SmartThings Smart Bulb, and a Schlage Connect Smart Deadbolt

[8]Available at https://github.com/akestoridis/zigator.
[9]Our changes have been submitted to Scapy's GitHub repository as a pull request.

Table 1: Number of different packet types in our dataset.

Packet Type	Number of Packets
MAC Acknowledgment	215,371
MAC Beacon	2,182
MAC Command	97,971
NWK Command	77,176
APS Acknowledgment	55,080
APS Command	3,317
ZDP Command	47,153
ZCL Command	73,259

(see Fig. 8h, 8c, 8d, and 8a respectively). The remaining devices are legacy Zigbee devices that use the default Trust Center link key for their commissioning process. In particular, the legacy Zigbee devices that we used are a Centralite 3-Series Smart Outlet, a SmartThings Motion Sensor (F-IRM-US-2), a SmartThings Multipurpose Sensor (F-MLT-US-2), and a Yale Assure Lock Touchscreen Deadbolt (see Fig. 8g, 8b, 8e, and 8f respectively).

In order to study a representative sample of NWK commands, we conducted eight experiments that we codenamed as H-T, where $H \in \{\texttt{sth2}, \texttt{sth3}\}$ and $T \in \{\texttt{room}, \texttt{duos}, \texttt{house}, \texttt{trios}\}$. We use $H = \texttt{sth2}$ to refer to the experiments where the STH-ETH-200 hub was used, while we use $H = \texttt{sth3}$ to refer to the experiments where the IM6001-V3P01 hub was used. We use $T = \texttt{room}$ to refer to the experiments where the devices were placed in a single room, while $T = \texttt{duos}$ refers to the experiments where the Zigbee network consisted of only a hub and one other device at a time. Furthermore, $T = \texttt{house}$ refers to the experiments where the devices were placed in three different rooms. Finally, $T = \texttt{trios}$ refers to the experiments where the Zigbee network consisted of only a hub, a Zigbee Router, and one other device at a time. For example, `sth3-room` corresponds to the experiment where we used the IM6001-V3P01 hub and all the devices were placed in a single room.

Before the start of each experiment, we performed a factory reset on the hub. Next, we registered the hub using the SmartThings smartphone application [28] to initialize our Zigbee network. Note that the default configuration for both hubs was to disable automatic firmware updates for Zigbee devices, while only the STH-ETH-200 hub accepted unsecured Rejoin Requests [34]. Our USRP N210 [14] was continuously capturing IEEE 802.15.4 packets on the selected channel. In order to collect a wide range of different NWK commands, we performed multiple actions during each experiment that included adding devices to our network, triggering sensors, issuing commands to actuators, reassociating devices, powering off devices, and causing PAN ID conflicts. We combined these actions to form a procedure that was common across our eight experiments. A detailed description of our procedure is included in our dataset, which will be available on CRAWDAD [11]. Table 1 shows the number of different packet types that we captured throughout our experiments, which lasted about 34.644 hours in total.

7 RESULTS

In this section we demonstrate the consequences of disabled MAC-layer security in centralized Zigbee networks. First, we show that valuable information can be inferred from operational Zigbee networks, which we then use to develop a decision tree for the identification of encrypted NWK commands. Finally, we launch selective jamming and spoofing attacks that force the end user to factory reset one of their devices and eventually expose the network key.

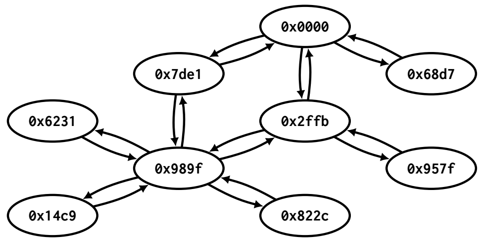


Figure 9: Example of an inferred topology from our dataset.

7.1 Reconnaissance Attacks

An attacker can infer the topology of a Zigbee network in their vicinity by logging distinct pairs of source and destination addresses from MAC headers. For example, Fig. 9 was generated by observing the addressing fields of MAC Data packets with PAN ID `0xd9b1` during the `sth2-house` experiment. While identifying the Zigbee Coordinator is trivial, since it always uses `0x0000` as its short address [45, p. 325], the topology alone is not sufficient to determine the logical device type of all the devices with absolute certainty.

Identifying logical device types actively. We can identify Zigbee Routers by exploiting the fact that only Zigbee Routers and the Zigbee Coordinator respond to Beacon Requests. By injecting a Beacon Request and then observing which devices responded with a beacon, the attacker can identify all Zigbee Routers that are within communication range. Recall that Zigbee devices do not utilize any security services on the MAC layer, meaning that an attacker can spoof any of the MAC commands shown in Table 2. To verify that the non-responding devices are indeed Zigbee End Devices, the attacker can spoof an Orphan Notification for a potential Zigbee End Device so that, if it is indeed a Zigbee End Device, its parent will respond with a Coordinator Realignment. We validated both of these attacks using our testbed without observing any side effects on the normal operation of the network. Note that during our experiments, even though only legacy Zigbee End Devices transmitted Orphan Notifications, all Zigbee Routers and Zigbee Coordinators responded with Coordinator Realignments regardless of whether the supposedly orphaned device was a legacy or a Zigbee 3.0 device.

Matching short and extended addresses. While the attacker does not have to impersonate another device to inject a valid Beacon Request, the attacker needs the device's extended address in order to successfully spoof an Orphan Notification. Nevertheless, the attacker can acquire this information by inspecting Zigbee traffic passively. Unlike the addressing fields of the MAC header (where either the short address, the extended address, or no address is used for the source and the destination fields of the packet [18, p. 57]), the addressing fields of the NWK header always include both short addresses and they may also include the corresponding extended addresses [45, p. 263]. In fact, every NWK command is required to include the extended address of its source [45, p. 269]. The extended

Table 2: Number of different MAC commands in our dataset.

MAC Command Name	Number of Packets
Association Request	223
Association Response	246
Data Request	95,198
Orphan Notification	121
Beacon Request	2,044
Coordinator Realignment	139

address of the destination is also included when it is known and the NWK command is not being broadcasted. In addition, every packet that utilizes NWK-layer security services is required to include the transmitter's extended address in the auxiliary header [45, p. 381], which will match with the short address of the source in the MAC header. Since MAC-layer security is disabled on Zigbee networks, all of these header fields are transmitted unencrypted. Thus, a passive attacker can match extended addresses with short addresses and vice versa simply by inspecting MAC Data packets.

Identifying logical device types passively. As we described at the beginning of this subsection, the Zigbee Coordinator can be identified without injecting any packet. An attacker would ideally like to be able to identify Zigbee Routers and Zigbee End Devices passively as well. This would require the observation of packets that are known to be transmitted (or received) either only by Zigbee Routers or only by Zigbee End Devices. Indeed, a Zigbee End Device will periodically transmit Data Requests to its parent in order to poll for pending packets [45, p. 201]. Caution is needed during the inspection of Data Requests because Zigbee Routers are also using them when they join the network [18, p. 32]; however, the source of the Data Request will use its extended address in this case [45, p. 488]. After a successful association, the source of the Data Request will use its short address to poll for pending packets [18, p. 71]. As we would expect, whenever the source address of a Data Request was a short address, it was transmitted by a Zigbee End Device throughout our experiments. Note that the attacker can use Data Requests to passively identify Zigbee Routers as well, since only Zigbee Routers and the Zigbee Coordinator are the recipients of these packets. However, in order to identify Zigbee Routers that do not have any children, the attacker will have to observe packets that are transmitted only by Zigbee Routers. There are MAC commands that could be used for this purpose, but they are not transmitted regularly. Hence, we turn to NWK commands (shown in Table 3). Indeed, the Zigbee Coordinator and all Zigbee Routers are periodically transmitting Link Status commands [45, p. 343]. Even though the NWK Command Identifier field is typically encrypted, an attacker can distinguish Link Status commands from other packets by inspecting some of their unencrypted header fields, as we show in Section 7.2. Thus, it is possible for a passive attacker to infer the logical device type of every Zigbee device.

Identifying hardware devices. The extended address of a device corresponds to a unique 64-bit IEEE address [18, p. 57], which contains an organizational identifier [20]. Since the Zigbee Alliance maintains a publicly available registry of certified Zigbee products [44], an attacker may attempt to identify the hardware device

Table 3: Number of different NWK commands in our dataset.

NWK Command Name	Number of Packets
Route Request	22,191
Route Reply	1,614
Network Status	10,389
Leave	340
Route Record	24,475
Rejoin Request	316
Rejoin Response	320
Link Status	16,305
Network Report	624
Network Update	371
End Device Timeout Request	141
End Device Timeout Response	90

from which the observed traffic is generated by using its organization name as a search keyword on that registry. In addition, smart hub vendors typically publish similar lists of supported devices (e.g., the "Works with SmartThings" webpage [29]), which can also be utilized if the vendor of the smart hub can be identified. The list of possible hardware devices can be further narrowed down by taking into account the logical device type that the attacker inferred for that device. For example, the extended address of the SmartThings Outlet (IM6001-OTP01) that we used in our experiments is `28:6d:97:00:01:09:4b:c8`, with its organizational identifier being `0x286d97`, which is assigned to SAMJIN Co., Ltd. [19]. At the time of writing, a search for that organization on the Zigbee Alliance's registry returns five certified Zigbee products. However, only one of these products is used as a Zigbee Router, which indeed matches our device. It is important to note that other organizational identifiers did not yield results that were as conclusive, because they corresponded to System-on-Chip (SoC) manufacturers whose products are found in a wide range of Zigbee devices.

Identifying legacy Zigbee devices. The attacker is interested in distinguishing legacy Zigbee devices from Zigbee 3.0 devices, especially in Zigbee 3.0 networks, because the former use the default Trust Center link key to join a network, which makes them potential targets for attacks as we explain in Section 7.3. The registry of certified Zigbee products provides this information, assuming that the attacker was able to identify the hardware device. For example, currently all certified Zigbee products by SAMJIN Co., Ltd. are Zigbee 3.0 devices. Additionally, we observed that in our experiments only Zigbee 3.0 devices transmitted End Device Timeout Requests. When this NWK command is transmitted, its parent is expected to transmit an End Device Timeout Response. However, this pair of NWK commands was introduced in the Zigbee PRO 2015 specification, so it is not supported by several legacy Zigbee devices [45, p. 363]. The attacker can make Zigbee End Devices rejoin their network by causing a PAN ID conflict, described in Section 7.3, in order to observe which ones transmitted an End Device Timeout Request and which parents responded with an End Device Timeout Response. As we show in the following subsection, the attacker can identify these NWK commands with 100% accuracy, even though their NWK Command Identifier field is transmitted encrypted.

Table 4: Sets of possible feature values for the identification of encrypted NWK commands. We denote logical device types with their initials, while the feature values that were observed in our dataset are in boldface. It should be noted that some of the feature values in the last two columns are not explicitly stated in the Zigbee PRO 2015 specification.

NWK Command Name	Payload Length (bytes)	Radius[†]	NWK Destination Type	NWK Source Type
Route Request	$\{\mathbf{5}, \mathbf{13}\}$	$\{\mathbf{2d}, \mathbf{2d-1}, \dots\}$	$\{\mathtt{\mathbf{0xfffc}}\}$	$\{\mathbf{ZC}, \mathbf{ZR}, \mathbf{ZED}\}$
Route Reply	$\{7, 15, \mathbf{23}\}$	$\{\mathbf{2d}, \mathbf{2d-1}, \dots\}$	$\{\mathbf{ZC}, \mathbf{ZR}\}$	$\{\mathbf{ZC}, \mathbf{ZR}\}$
Network Status	$\{1, \mathbf{3}\}$	$\{\mathbf{2d}, \mathbf{2d-1}, \dots\}$	$\{\mathbf{ZC}, \mathbf{ZR}, \mathbf{ZED}, \mathtt{0xfffd}\}$	$\{\mathrm{ZC}, \mathbf{ZR}, \mathrm{ZED}\}$
Leave	$\{\mathbf{1}\}$	$\{\mathbf{1}\}$	$\{\mathbf{ZC}, \mathbf{ZR}, \mathbf{ZED}, \mathtt{\mathbf{0xfffd}}\}$	$\{\mathbf{ZC}, \mathbf{ZR}, \mathbf{ZED}\}$
Route Record	$\{\mathbf{1}, \mathbf{3}, \mathbf{5}, \dots\}$	$\{\mathbf{2d}, \mathbf{2d-1}, \dots\}$	$\{\mathbf{ZC}, \mathrm{ZR}\}$	$\{\mathrm{ZC}, \mathbf{ZR}, \mathbf{ZED}\}$
Rejoin Request	$\{\mathbf{1}\}$	$\{\mathbf{1}\}$	$\{\mathbf{ZC}, \mathbf{ZR}\}$	$\{\mathbf{ZR}, \mathbf{ZED}\}$
Rejoin Response	$\{\mathbf{3}\}$	$\{\mathbf{1}\}$	$\{\mathbf{ZR}, \mathbf{ZED}\}$	$\{\mathbf{ZC}, \mathbf{ZR}\}$
Link Status	$\{\mathbf{1}, \mathbf{4}, \mathbf{7}, \dots\}$	$\{\mathbf{1}\}$	$\{\mathtt{\mathbf{0xfffc}}\}$	$\{\mathbf{ZC}, \mathbf{ZR}\}$
Network Report	$\{\mathbf{11}, 13, 15, \dots\}$	$\{\mathbf{2d}, \mathbf{2d-1}, \dots\}$	$\{\mathbf{ZC}\}$[‡]	$\{\mathbf{ZR}\}$[‡]
Network Update	$\{\mathbf{12}\}$	$\{\mathbf{2d}, \mathbf{2d-1}, \dots\}$	$\{\mathtt{\mathbf{0xffff}}\}$	$\{\mathbf{ZC}\}$[‡]
End Device Timeout Request	$\{\mathbf{2}\}$	$\{\mathbf{1}\}$	$\{\mathbf{ZC}, \mathbf{ZR}\}$	$\{\mathbf{ZED}\}$
End Device Timeout Response	$\{\mathbf{2}\}$	$\{\mathbf{1}\}$	$\{\mathbf{ZED}\}$	$\{\mathbf{ZC}, \mathbf{ZR}\}$

[†]The radius is the number of hops that a Zigbee packet is allowed to make [45, p. 221], while we use d to denote the maximum depth of the Zigbee network.
[‡]We assume that the Zigbee Coordinator has been designated as the network manager, which is the default configuration [45, p. 297].

7.2 Identification of NWK Commands

It should be clear that the attacker would benefit significantly from having the ability to identify encrypted NWK commands. While the NWK Frame Type field is always transmitted unencrypted, since it is part of the NWK header and MAC-layer security is disabled, the NWK Command Identifier field is typically encrypted because it is part of the NWK payload. Hence, a passive observer can distinguish NWK commands from other packet types, but not which NWK command it actually is. We constructed Table 4, which contains sets of possible values for four features that could be used for the classification of encrypted NWK commands. The values of the Payload Length and Radius columns were based on the requirements of the Zigbee PRO 2015 specification [45, p. 269]. However, the specification does not explicitly state the possible types of the NWK destination and source address fields for all NWK commands. As a result, some of the values in these two columns are based on our understanding of the specification, the experiments that we conducted, and the assumptions that we made in Section 3. Notice that the payload length varies significantly between different NWK commands, while half of them are limited to single-hop transmissions. Furthermore, notice that certain NWK commands are limited to only certain NWK destination and source types.

Since the length of the MAC protocol data unit (MPDU) is included in the PHY header [18, p. 160] and all header fields that affect its expected length are transmitted unencrypted (e.g., whether the extended address of the NWK destination is present or not), the attacker can compute the payload length of NWK commands as they are being transmitted. The Radius field, which indicates how many hops a packet can perform, is transmitted unencrypted as well. Note that even though the Radius field of multi-hop commands may reach a value of 1, we can still distinguish them from single-hop commands by comparing the addressing fields in the MAC and NWK headers to determine whether the packet has traversed the network or not. The NWK destination and source types can be inferred using the methods that we presented in Section 7.1. Using just these features we can construct a decision tree that can identify 6 out of the 12 possible NWK commands with 100% accuracy. The remaining NWK commands may reach an impure leaf node that represents either two or three possibilities for their identification.

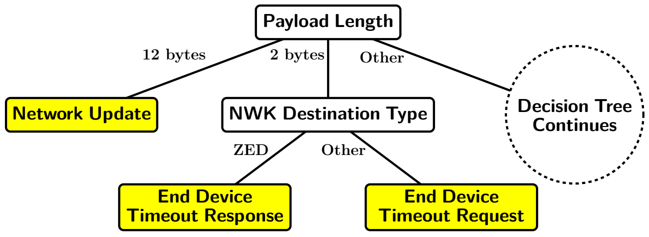


Figure 10: Partial depiction of our decision tree that shows how to identify three NWK commands with 100% accuracy.

We now describe our derived decision rules that achieve 100% accuracy[10]. From Table 4, we observe that only End Device Timeout Request and Response commands can have a payload length of 2 bytes. Furthermore, the source of an End Device Timeout Request command is always a Zigbee End Device and the destination is either the Zigbee Coordinator or a Zigbee Router, while this pattern is reversed for an End Device Timeout Response command. Thus, we can distinguish these two NWK commands from each other by taking into account the NWK destination type. This is depicted in the partial decision tree that we provide in Fig. 10. Next, as we can see in Table 4, the Link Status and the Route Request commands are the only NWK commands that are being broadcasted to all Zigbee Routers and the Zigbee Coordinator (denoted by the `0xfffc` broadcast address [45, p. 354]). These can be distinguished based on whether they are a single-hop NWK command or not. In addition, notice that Link Status commands are being transmitted only by Zigbee Routers and the Zigbee Coordinator, but we did not have to rely on this fact in order to identify them as Link Status commands.

[10]The remaining rules can be derived from the decision tree shown in Appendix A.

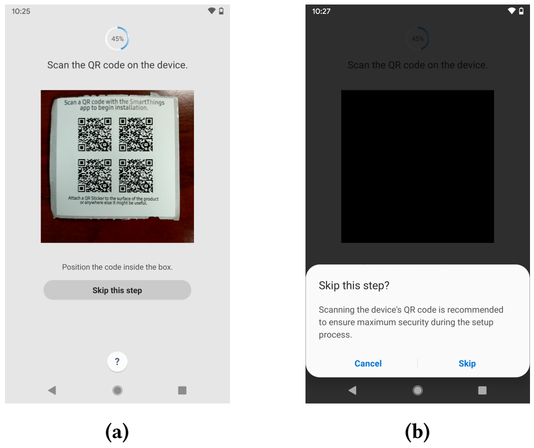
Thus, as we mentioned in Section 7.1, a passive attacker can identify Zigbee Routers through the detection of Link Status commands. We can also identify Network Update and Rejoin Response commands with 100% accuracy, which we discuss in the following subsection in the context of the attacks that we launched.

7.3 Key-Transport Attacks

Security researchers have been aware that the transportation of the network key to new devices may not be sufficiently protected since early versions of the Zigbee protocol [39], resulting in "a brief moment of vulnerability" [45, p. 376]. Regarding legacy Zigbee networks, Zillner and Strobl suggested that an attacker could trick the end user to factory reset one of their Zigbee devices using jamming techniques, essentially extending the time frame of the vulnerability [49]. However, a Zigbee 3.0 device can join a Zigbee 3.0 network using an install code, which is used to derive a preconfigured Trust Center link key that is device-specific and transferred to the Trust Center over an out-of-band communication channel in order to encrypt the Zigbee packet that will transport the network key to the new device [46, p. 73]. We now study the effectiveness of this strategy in overcoming the network key leakage vulnerability.

Assessing the commissioning of Zigbee 3.0 devices. The install codes of the Zigbee 3.0 devices that we purchased were found on the devices and inside their boxes, while additional stickers were provided for the light bulb. To add a Zigbee 3.0 device to our Zigbee 3.0 network, we scanned its QR code with our smartphone, as shown in Fig. 11a. An option to add a Zigbee 3.0 device without scanning its QR code was provided, which displayed the message shown in Fig. 11b when selected. In that case, the Zigbee Coordinator used the default Trust Center link key to encrypt the Transport Key command. Assuming that unsecured Rejoin Requests are not accepted [34], the attacker's main strategy for obtaining the network key is to launch a denial-of-service attack that would force the end user to factory reset a device that uses a known Trust Center link key to join the network. The attacker could approach this task by identifying and targeting legacy Zigbee devices. Alternatively, the attacker could target a Zigbee 3.0 device if they can gain access to its install code, which could be leaked if its QR code is visible to the attacker, if the end user did not dispose the device's box securely, or if the device was previously owned by a different user. The QR codes of our Zigbee 3.0 devices also contained the extended address of their corresponding devices, which the attacker could then easily identify using the methods described in Section 7.1. If the attacker was able to gain access to an install code, they may be more likely to obtain the network key by targeting only the corresponding Zigbee 3.0 device because the end user may not consider the possibility of its install code being leaked and factory reset it without hesitation.

Forcing the factory reset of a device. The attacker would ideally like to minimize the number of packets that they have to jam in order to force the factory reset of a device. We realize that this can be achieved by causing PAN ID conflicts. The injection of a forged beacon, that uses the same PAN ID as the end user's Zigbee network but with a different Extended PAN ID (EPID)[11], is sufficient to initiate the PAN ID conflict resolution process. Each Zigbee Router that received the forged beacon informs the network manager, which is typically the Zigbee Coordinator [45, p. 297], with the Network Report command. The network manager then selects a new PAN ID and broadcasts it to all devices in the network with the Network Update command [45, p. 333]. Shortly after the transmission of the Network Update command, the network manager and all the devices that received it switch to the new PAN ID. Zigbee End Devices may not receive the Network Update command before the PAN ID changes, in which case they try to rejoin the network [45, p. 450]. Thus, if the attacker selectively jams the Network Update commands issued by the network manager, some devices will not switch to the new PAN ID automatically. As we mentioned in Section 7.2, the attacker can identify Network Update commands because they are the only NWK commands that can have a payload length of 12 bytes. Rejoin Responses can also be identified because they are the only NWK commands that can have a payload length of 3 bytes and are limited to a single-hop transmission. However, during our experiments we observed that even if we jam all Rejoin Responses, some devices were still able to rejoin the network with the new PAN ID. To keep a device disconnected, we had to selectively jam the updated beacons to prevent the device from updating its PAN ID. We also observed that Zigbee End Devices did not initiate the rejoin process as expected if we spoofed MAC acknowledgments for their Data Requests after a PAN ID conflict. Launching these denial-of-service attacks leads to the following outcomes: (a) the end user will be unable to change the state of their affected actuators, (b) if an affected sensor detects an event it will be unable to inform the end user, and (c) automation rules that depend on affected devices will not function properly. If the attacker successfully targets a Zigbee device that uses a known Trust Center link key and the end user decides to factory reset it, then the attacker can gain access to the network key by sniffing and decrypting the Transport Key command during its commissioning process. Then, the attacker can decrypt most of the encrypted payloads and inject commands that change the state of the end user's devices, including Zigbee 3.0 devices that use install codes.


10:25
45%
Scan the QR code on the device.
Scan a QR code with the SmartThings app to begin installation.
Attach a QR Sticker to the surface of the product or anywhere else it might be useful.
Position the code inside the box.
Skip this step

10:27
Skip this step?
Scanning the device's QR code is recommended to ensure maximum security during the setup process.

Figure 11: Screenshots from the SmartThings smartphone application [28] that show (a) the commissioning of a Zigbee 3.0 device by scanning its QR code and (b) the warning message that is displayed when a user tries to skip this step.

[11] Zigbee devices transmit the EPID of their network in their beacons unencrypted.

Unexpected behaviors. We launched the aforementioned attacks by modifying the firmware of our ATUSB [25]. We observed some unexpected behaviors from our Zigbee Routers when they failed to receive Network Update commands. While the SmartThings Outlet (IM6001-OTP01) initiated the rejoin process relatively shortly after it determined that it cannot reach the Zigbee Coordinator anymore, the Centralite 3-Series Smart Outlet initiated that process after about 25 minutes. This behavior can also be observed in our dataset. More interestingly, the SmartThings Smart Bulb did not initiate the rejoin process like the smart outlets for up to the 38 hours that we monitored it. This seems to be a firmware issue since the Centralite outlet is a legacy Zigbee device while the SmartThings bulb is a Zigbee 3.0 device. We note that such behavior benefits the attacker because they will have to jam less packets than expected. Given the large number of certified Zigbee devices, it is unknown how many do not initiate or significantly delay the rejoin process when they fail to receive the Network Update command.

Responsible disclosure. We reported our findings to the Zigbee Alliance, whose validity was confirmed by one of their representatives. They stated that they were aware of PAN ID conflict attacks and that they have implemented specification changes that will prevent malicious PAN ID changes. Regarding our selective jamming of Network Update commands, they commented that PAN ID changes may be missed even in non-malicious scenarios and that, as a result, future specifications will require a more aggressive algorithm. They also argued that it is difficult for the network key to be leaked from Zigbee 3.0 devices. However, since legacy Zigbee devices are still available on the market and consumers may not have replaced all of them with Zigbee 3.0 devices, we argue that key-transport attacks can still be launched in the wild.

Security enhancements. We argue that the security of the commissioning process would improve significantly if the Trust Center link key was *reconfigurable* over an out-of-band communication channel. In particular, this encryption key could be different whenever a device tries to join a new network and be transported over NFC. Given that smart home ecosystems like SmartThings already rely on a smartphone application for users to add devices to their Zigbee networks, the use of NFC should not impact usability in any significant way. However, the manufacturing cost of the devices may increase due to the additional hardware required. Finally, the end users should be made aware of the security risks that the use of a legacy Zigbee device would introduce to their networks.

8 RELATED WORK

Several researchers have demonstrated information leakage from Zigbee-enabled smart homes. Acar et al. inferred user activities by creating traffic profiles and relying on traffic rate variations to detect device state changes [1]. Zhang et al. extracted fingerprints for several event types in order to detect misbehaving smart home applications [41]. Trimananda et al. were able to infer events related to Zigbee devices by identifying packet-level signatures from the Wi-Fi and Ethernet traffic that the smartphone and smart hub generated in order to communicate with cloud servers [35]. Gu et al. extracted event fingerprints in order to detect anomalies in smart home applications and discover hidden vulnerabilities [17]. A common characteristic in these studies is that they followed protocol-agnostic approaches in order to be applicable to a wide range of smart home devices. In contrast, we focus on Zigbee devices and rely on the inspection of multiple header fields in order to infer information that enables us to launch disruptive attacks.

Zillner and Strobl argued that an attacker can gain access to the network key of a centralized legacy Zigbee network by utilizing jamming techniques in order to trick the end user to factory reset one of their Zigbee devices [49]. However, their demonstration included only a constant jammer, whose detection is relatively easy [40]. Alternatively, an attacker could jam packets reactively. Wilhelm et al. implemented a reactive jammer by programming the FPGA of a software-defined radio [36], while Wood et al. implemented one on the microcontroller of an IEEE 802.15.4 device [38]. Nevertheless, more sophisticated attacks are made feasible if the attacker can jam packets selectively. Bloessl demonstrated that low-cost selective jammers can be realized by programming a microcontroller with an IEEE 802.15.4 transceiver [6]. While Bloessl's implementation selectively jams all packets destined for a specific device, the implementation that we present supports the definition of arbitrary jamming conditions. Our implementation enables us to selectively jam packets and then inject forged MAC acknowledgments, as Sastry and Wagner described [30]. We show that, due to the lack of MAC-layer security in Zigbee networks, an attacker can easily identify and selectively jam certain NWK commands.

Goodspeed et al. extended Scapy to dissect and forge several packet types of IEEE 802.15.4-based networks [16]. Support for more packet types was added over time, which enabled researchers to launch command injection attacks against centralized Zigbee networks when the network key is known [8, 21, 49]. We further extend Scapy's capabilities to enable detailed analysis of Zigbee packets. Knight demonstrated injection of forged packets to unlock a door lock after the jamming of Data Requests [21]. In this work we explore how an attacker can obtain the network key, which would enable such command injection attacks. Brown and Gleason caused PAN ID conflicts and then spoofed MAC acknowledgments so that a door sensor would not be able to notify the alarm system if its door was opened [8]. We repurpose this attack to prevent a targeted Zigbee End Device from updating its PAN ID until the end user decides to factory reset it and expose the network key. In addition, our decision tree enables us to disconnect Zigbee Routers as well by selectively jamming Network Update commands.

9 CONCLUSION

In this work we introduce Zigator and a testbed design that enable in-depth security analysis of Zigbee networks. We delineate multiple reconnaissance attacks against centralized Zigbee networks and show that, despite NWK-layer encryption, an attacker can identify certain NWK commands with 100% accuracy. We use this information to develop selective jamming and spoofing attacks that can lead to the exposure of the network key by targeting devices that use known Trust Center link keys. We provide recommendations to protect against leakage of the network key, while we publicly release our software tools and dataset. Our work sheds light on the consequences of disabling MAC-layer security in centralized Zigbee networks. As we demonstrate, NWK-layer security is insufficient to protect against several passive and active attacks.

REFERENCES

[1] Abbas Acar, Hossein Fereidooni, Tigist Abera, Amit Kumar Sikder, Markus Miettinen, Hidayet Aksu, Mauro Conti, Ahmad-Reza Sadeghi, and A. Selcuk Uluagac. 2018. Peek-a-Boo: I see your smart home activities, even encrypted! arXiv:1808.02741 [cs.CR]

[2] Atmel Corporation. 2009. *AT86RF231/ZU/ZF datasheet.* 8111C-MCU Wireless-09/09.

[3] Luigi Atzori, Antonio Iera, and Giacomo Morabito. 2010. The Internet of Things: A survey. *Computer Networks* 54, 15 (2010), 2787–2805. https://doi.org/10.1016/j.comnet.2010.05.010

[4] Bastian Bloessl. [n.d.]. *gr-foo.* Retrieved May 15, 2020 from https://github.com/bastibl/gr-foo

[5] Bastian Bloessl. [n.d.]. *gr-ieee802-15-4.* Retrieved May 15, 2020 from https://github.com/bastibl/gr-ieee802-15-4

[6] Bastian Bloessl. 2019. *Low-Cost ZigBee Selective Jamming.* Retrieved May 15, 2020 from https://www.bastibl.net/reactive-zigbee-jamming/

[7] Bastian Bloessl, Christoph Leitner, Falko Dressler, and Christoph Sommer. 2013. A GNU Radio-based IEEE 802.15.4 Testbed. In *Proceedings of the 12th GI/ITG KuVS Fachgespräch "Drahtlose Sensornetze" (FGSN).* 37–40.

[8] Francis Brown and Matthew Gleason. 2019. ZigBee Hacking: Smarter Home Invasion with ZigDiggity. Presented at Black Hat USA 2019.

[9] Ed Callaway, Paul Gorday, Lance Hester, Jose A. Gutierrez, Marco Naeve, Bob Heile, and Venkat Bahl. 2002. Home Networking with IEEE 802.15.4: A Developing Standard for Low-Rate Wireless Personal Area Networks. *IEEE Communications Magazine* 40, 8 (2002), 70–77. https://doi.org/10.1109/MCOM.2002.1024418

[10] Check Point Software Technologies Ltd. 2020. *The Dark Side of Smart Lighting: Check Point Research Shows How Business and Home Networks Can Be Hacked from a Lightbulb.* Retrieved May 15, 2020 from https://blog.checkpoint.com/2020/02/05/the-dark-side-of-smart-lighting-check-point-research-shows-how-business-and-home-networks-can-be-hacked-from-a-lightbulb/

[11] CRAWDAD. [n.d.]. *CRAWDAD: A Community Resource for Archiving Wireless Data At Dartmouth.* Retrieved May 15, 2020 from https://crawdad.org/

[12] Morris Dworkin. 2004. *Recommendation for Block Cipher Modes of Operation: The CCM Mode for Authentication and Confidentiality.* https://doi.org/10.6028/NIST.SP.800-38C NIST Special Publication 800-38C.

[13] Helder Eijs. [n.d.]. *PyCryptodome: A self-contained cryptographic library for Python.* Retrieved May 15, 2020 from https://github.com/Legrandin/pycryptodome

[14] Ettus Research. [n.d.]. *USRP N210 Software Defined Radio (SDR).* Retrieved May 15, 2020 from https://www.ettus.com/all-products/un210-kit/

[15] GNU Radio. [n.d.]. *GNU Radio – the Free and Open Software Radio Ecosystem.* Retrieved May 15, 2020 from https://github.com/gnuradio/gnuradio

[16] Travis Goodspeed, Sergey Bratus, Ricky Melgares, Ryan Speers, and Sean W. Smith. 2012. Api-do: Tools for Exploring the Wireless Attack Surface in Smart Meters. In *Proceedings of the 45th Hawaii International Conference on System Sciences (HICSS).* 2133–2140. https://doi.org/10.1109/HICSS.2012.115

[17] Tianbo Gu, Zheng Fang, Allaukik Abhishek, Hao Fu, Pengfei Hu, and Prasant Mohapatra. 2020. IoTGAZE: IoT Security Enforcement via Wireless Context Analysis. (2020). To appear in the Proceedings of the 2020 IEEE Conference on Computer Communications (INFOCOM).

[18] IEEE Computer Society. 2011. *IEEE Standard for Local and metropolitan area networks–Part 15.4: Low-Rate Wireless Personal Area Networks (LR-WPANs).* https://doi.org/10.1109/IEEESTD.2011.6012487 IEEE Std 802.15.4-2011.

[19] IEEE Registration Authority. [n.d.]. *MAC Address Block Large (MA-L).* Retrieved May 15, 2020 from http://standards-oui.ieee.org/oui/oui.txt

[20] IEEE Registration Authority. 2019. *Guidelines for Use of Extended Unique Identifier (EUI), Organizationally Unique Identifier (OUI), and Company ID (CID).* Retrieved May 15, 2020 from https://standards.ieee.org/content/dam/ieee-standards/standards/web/documents/tutorials/eui.pdf

[21] Matt Knight. 2016. Wireless Lockpicking: Exploring 802.15.4 Command Injection. Presented at Cyberspectrum 14.

[22] Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone. 1997. Hash functions based on block ciphers. In *Handbook of Applied Cryptography.* CRC Press.

[23] Philipp Morgner, Stephan Mattejat, Zinaida Benenson, Christian Müller, and Frederik Armknecht. 2017. Insecure to the Touch: Attacking ZigBee 3.0 via Touchlink Commissioning. In *Proceedings of the 10th ACM Conference on Security and Privacy in Wireless and Mobile Networks (WiSec).* 230–240. https://doi.org/10.1145/3098243.3098254

[24] National Institute of Standards and Technology. 2001. *Advanced Encryption Standard (AES).* https://doi.org/10.6028/NIST.FIPS.197 FIPS 197.

[25] Qi Hardware Inc. [n.d.]. *Ben-WPAN Overview.* Retrieved May 15, 2020 from http://downloads.qi-hardware.com/people/werner/wpan/web/

[26] Qi Hardware Inc. [n.d.]. *IEEE 802.15.4 subsystem.* Retrieved May 15, 2020 from http://projects.qi-hardware.com/index.php/p/ben-wpan/

[27] Eyal Ronen, Colin O'Flynn, Adi Shamir, and Achi-Or Weingarten. 2017. IoT Goes Nuclear: Creating a ZigBee Chain Reaction. In *Proceedings of the 2017 IEEE Symposium on Security and Privacy (SP).* 195–212. https://doi.org/10.1109/SP.2017.14

[28] Samsung Electronics Co., Ltd. [n.d.]. *SmartThings.* Retrieved February 27, 2020 from https://play.google.com/store/apps/details?id=com.samsung.android.oneconnect

[29] Samsung Electronics Co., Ltd. [n.d.]. *Works With SmartThings.* Retrieved May 15, 2020 from https://www.smartthings.com/products

[30] Naveen Sastry and David Wagner. 2004. Security Considerations for IEEE 802.15.4 Networks. In *Proceedings of the 3rd ACM Workshop on Wireless Security (WiSe).* 32–42. https://doi.org/10.1145/1023646.1023654

[31] Thomas Schmid, Oussama Sekkat, and Mani B. Srivastava. 2007. An Experimental Study of Network Performance Impact of Increased Latency in Software Defined Radios. In *Proceedings of the Second ACM International Workshop on Wireless Network Testbeds, Experimental Evaluation and Characterization (WinTECH).* 59–66. https://doi.org/10.1145/1287767.1287779

[32] Scikit-learn Project. [n.d.]. *Scikit-learn: Machine Learning in Python.* Retrieved May 15, 2020 from https://github.com/scikit-learn/scikit-learn

[33] SecDev. [n.d.]. *Scapy: the Python-based interactive packet manipulation program & library.* Retrieved May 15, 2020 from https://github.com/secdev/scapy

[34] SmartThings Community. 2015. *Security of SmartThings ecosystem.* Retrieved May 15, 2020 from https://community.smartthings.com/t/security-of-smartthings-ecosystem/30827/5

[35] Rahmadi Trimananda, Janus Varmarken, Athina Markopoulou, and Brian Demsky. 2020. Packet-Level Signatures for Smart Home Devices. In *Proceedings of the 2020 Network and Distributed System Security Symposium (NDSS).* https://doi.org/10.14722/ndss.2020.24097

[36] Matthias Wilhelm, Ivan Martinovic, Jens B. Schmitt, and Vincent Lenders. 2011. Short Paper: Reactive Jamming in Wireless Networks: How Realistic is the Threat?. In *Proceedings of the 4th ACM Conference on Wireless Network Security (WiSec).* 47–52. https://doi.org/10.1145/1998412.1998422

[37] Wireshark Foundation. [n.d.]. *The Wireshark network protocol analyzer.* Retrieved May 15, 2020 from https://code.wireshark.org/review/gitweb?p=wireshark.git

[38] Anthony D. Wood, John A. Stankovic, and Gang Zhou. 2007. DEEJAM: Defeating Energy-Efficient Jamming in IEEE 802.15.4-based Wireless Networks. In *Proceedings of the 4th Annual IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks (SECON).* 60–69. https://doi.org/10.1109/SAHCN.2007.4292818

[39] Joshua Wright and Johnny Cache. 2015. Hacking ZigBee. In *Hacking Exposed Wireless: Wireless Security Secrets & Solutions* (3rd ed.). McGraw-Hill Education Group.

[40] Wenyuan Xu, Wade Trappe, Yanyong Zhang, and Timothy Wood. 2005. The Feasibility of Launching and Detecting Jamming Attacks in Wireless Networks. In *Proceedings of the 6th ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc).* 46–57. https://doi.org/10.1145/1062689.1062697

[41] Wei Zhang, Yan Meng, Yugeng Liu, Xiaokuan Zhang, Yinqian Zhang, and Haojin Zhu. 2018. HoMonit: Monitoring Smart Home Apps from Encrypted Traffic. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security (CCS).* 1074–1088. https://doi.org/10.1145/3243734.3243820

[42] Zigbee Alliance. [n.d.]. *Smart Home.* Retrieved May 15, 2020 from https://zigbeealliance.org/market-uses/smart-home/

[43] Zigbee Alliance. [n.d.]. *Zigbee.* Retrieved May 15, 2020 from https://zigbeealliance.org/solution/zigbee/

[44] Zigbee Alliance. [n.d.]. *Zigbee Certified Products Archives.* Retrieved May 15, 2020 from https://zigbeealliance.org/product_type/certified_product/

[45] Zigbee Alliance. 2015. *ZigBee Specification.* ZigBee Document 05-3474-21.

[46] Zigbee Alliance. 2016. *Base Device Behavior Specification.* ZigBee Document 13-0402-13.

[47] Zigbee Alliance. 2016. *ZigBee Cluster Library Specification.* ZigBee Document 07-5123.

[48] Zigbee Alliance. 2017. *zigbee: Securing the Wireless IoT.* White Paper.

[49] Tobias Zillner and Sebastian Strobl. 2015. ZigBee Exploited - The Good, the Bad and the Ugly. Presented at Black Hat USA 2015.

A DECISION TREE FOR THE IDENTIFICATION OF NWK COMMANDS

In Section 7.2 we presented a partial depiction of our decision tree. The full depiction of our decision tree is shown in Fig. 12. The relative frequency by which each encrypted NWK command in our dataset reached a pure leaf node of our decision tree is provided in Table 5. As we explained in Section 7.2, half of them always reach a pure leaf node. Regarding the remaining half, their varying

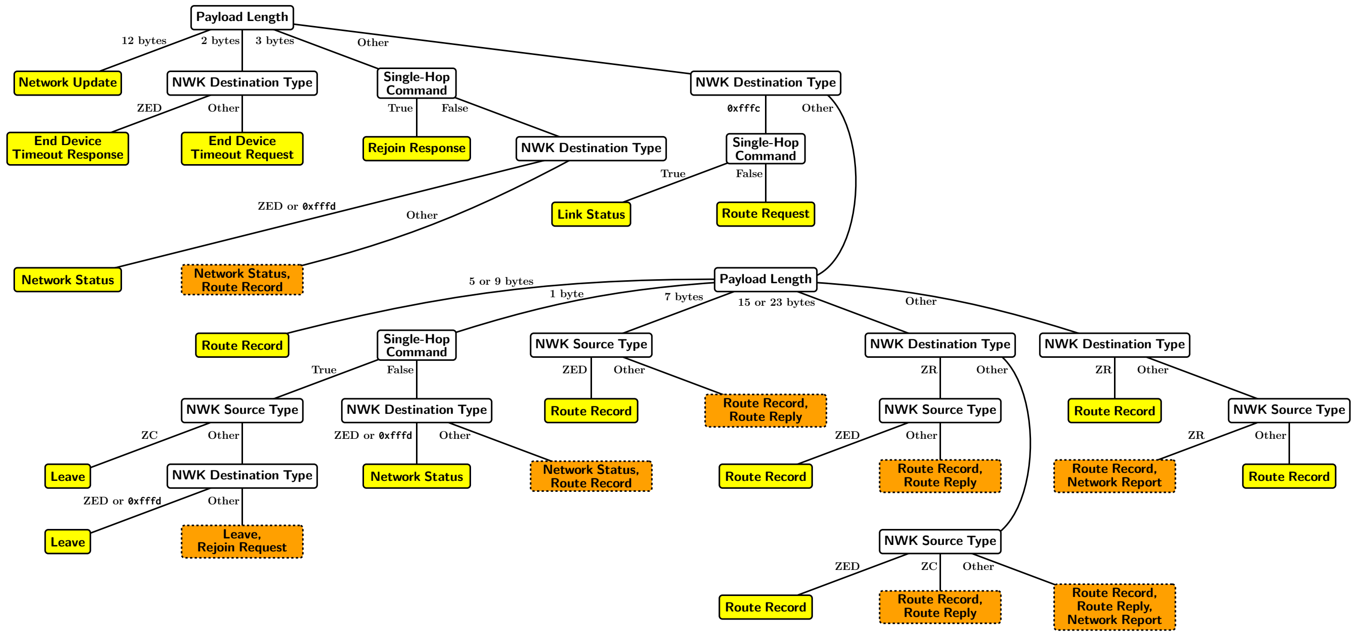


Figure 12: The decision tree that we developed for the identification of encrypted NWK commands based on the assumptions that we stated in Sections 3 and 7.2 as well as the reconnaissance attacks that we presented in Section 7.1. Pure leaf nodes are colored yellow with a solid outline, while impure leaf nodes are colored orange with a dashed outline.

percentages indicate that there are several cases where they reach an impure leaf node of our decision tree. However, even though the utilized features were not sufficient to always identify this set of encrypted NWK commands uniquely, their identification has been narrowed down to either two or three possibilities. Furthermore, it is important to note that while our dataset does not contain all possible variations of the 12 possible NWK commands, our decision tree was developed using information from the Zigbee PRO 2015 specification itself. For example, there may be cases where the payload length of a Network Status command is 1 byte [45, p. 278], although we did not observe any during our experiments, which resulted in the creation of an impure leaf node that is reached by a large number of Route Record commands that we did observe.

As another example, consider an encrypted NWK command where its payload length is 1 byte, its NWK destination is not the `0xfffc` broadcast address, and it is limited to a single-hop transmission. We can observe from Table 4 that only two NWK commands can have such values: the Leave command and the Rejoin Request command. If its NWK source is the Zigbee Coordinator, then we can conclude with certainty that it is a Leave command because Zigbee Coordinators do not transmit Rejoin Request commands. Similarly, if the NWK destination is a Zigbee End Device or the `0xfffd` broadcast address, then the aforementioned NWK command must be a Leave command. However, if the NWK source is a Zigbee End Device and the NWK destination is the Zigbee Coordinator, then we cannot differentiate a Leave command from a Rejoin Request command without having to rely on additional features. In addition, consider an encrypted NWK command where its payload length is 23 bytes, its NWK destination is the Zigbee Coordinator, and its NWK source is a Zigbee Router. This NWK command could be (a) a Route Record command that describes a relay path that consists of 11 hops, (b) a Route Reply command that includes the extended addresses of its originator and responder, or (c) a Network Report command that contains the PAN IDs of 7 neighboring networks. Depending on the topology of the inspected Zigbee network and previously observed events, one of these possibilities may be more likely than the others. We consider the development of additional features for our decision tree as part of our future work.

Table 5: Percentage of encrypted NWK commands in our dataset that reached a pure leaf node of our decision tree.

NWK Command Name	Percentage of Packets
Route Request	100.0%
Route Reply	0.0%
Network Status	0.4%
Leave	72.1%
Route Record	4.6%
Rejoin Request	0.0%
Rejoin Response	100.0%
Link Status	100.0%
Network Report	0.0%
Network Update	100.0%
End Device Timeout Request	100.0%
End Device Timeout Response	100.0%