On the Security of Thread Networks: Experimentation with OpenThread-Enabled Devices

Dimitrios-Georgios Akestoridis
Carnegie Mellon University
Pittsburgh, Pennsylvania, USA
akestoridis@cmu.edu

Vyas Sekar
Carnegie Mellon University
Pittsburgh, Pennsylvania, USA
vsekar@andrew.cmu.edu

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

ABSTRACT

The Thread networking protocol is expected to be utilized by a plethora of smart home devices as one of the IP-based networking technologies that will be supported by the Matter standard that is being developed by members of the Connectivity Standards Alliance. Thread has been developed by the Thread Group as an application-agnostic protocol that builds on top of the IEEE 802.15.4 standard to enable IPv6-based low-power wireless mesh networking. However, unlike other IEEE 802.15.4-based protocols like Zigbee, the security of Thread networks has been relatively less analyzed in the literature. Given that commercial Thread devices are expected to interact with the physical world, vulnerabilities in their communication protocols could impact the physical security of end users. In this work we analyze the security of Thread networks by repurposing hardware and software tools that have been used for the security analysis of Zigbee networks. We used development boards that were flashed with OpenThread binaries to gain insight into the nature of Thread traffic and to study their susceptibility to a set of energy depletion attacks and online password guessing attacks. Lastly, we are publicly releasing our software enhancements as well as our dataset of captured Thread packets.

CCS CONCEPTS

• **Security and privacy → Mobile and wireless security**; *Denial-of-service attacks*; • **Networks → Mobile and wireless security**; *Denial-of-service attacks*; *Home networks*; *Sensor networks.*

KEYWORDS

Thread, IEEE 802.15.4, energy depletion attacks, online password guessing attacks, OpenThread

ACM Reference Format:

Dimitrios-Georgios Akestoridis, Vyas Sekar, and Patrick Tague. 2022. On the Security of Thread Networks: Experimentation with OpenThread-Enabled Devices. In *WiSec '22: 15th ACM Conference on Security and Privacy in Wireless and Mobile Networks, May 16–May 19, 2022, San Antonio, Texas, USA.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/xxxxxxx.xxxxxxx

WiSec '22, May 16–May 19, 2022, San Antonio, Texas, USA

ACM ISBN 978-1-4503-XXXX-X/21/06…$15.00
https://doi.org/10.1145/xxxxxxx.xxxxxxx

1 INTRODUCTION

One of the main challenges of the Internet of Things (IoT) [8] is to make everyday devices accessible remotely and securely as either sensors, actuators, or both. Door locks, motion sensors, cameras, thermostats, speakers, and light bulbs are only a few examples of such devices that are now connected to the Internet. End users can monitor the state of their IoT devices (e.g., whether a door is locked or unlocked), issue commands to change the state of their IoT devices (e.g., unlock a door), and automate the execution of certain tasks (e.g., turning the lights on whenever motion is detected). However, different IoT devices may have different requirements. For example, a smart speaker may require a high-data-rate connection, while a smart motion sensor may require a long battery life. As a result, a variety of communication protocols may be utilized in order to support the operation of a smart environment.

Amazon, Apple, Google, and the Zigbee Alliance (now known as the Connectivity Standards Alliance) announced the formation of a new working group in December 2019 for the development of an IP-based smart home connectivity standard that would increase compatibility among IoT devices [15]. This unifying standard was initially referred to as Project Connected Home over IP (CHIP), which was rebranded as Matter in May 2021 [17]. At the time of writing, the launch of Matter 1.0 is expected to take place in Fall 2022, which will be followed by the opening of the formal certification program for Matter-enabled products [16]. As illustrated in Figure 1, Thread [49] is currently the only IEEE 802.15.4-based networking protocol that has been selected for the Matter standard [18]. More specifically, Thread has been developed by the Thread Group as an application-agnostic protocol that enables IPv6-based low-power wireless mesh networking [32]. Furthermore, OpenThread [43] is an open-source implementation of the Thread networking protocol that has been ported to multiple platforms. It should be noted that, since Thread is application-layer agnostic, manufacturers of Thread products have the flexibility to choose from a variety of application layers to enable device connectivity across different networks, with the Matter application layer expected to be one such option.

Similar to other IoT devices that interact with the physical world, vulnerabilities in Thread devices can affect the physical security of end users. However, the security of Thread networks has not been popular in the literature. This motivated us to study whether Thread is susceptible to known attacks on Zigbee [19], an IEEE 802.15.4-based protocol that has been used in smart environments for several years, as well as attacks that are unique to Thread. Our contributions in this paper can be summarized as follows:

- We describe a set of observations about the nature of Thread traffic, which we made by performing multiple experiments

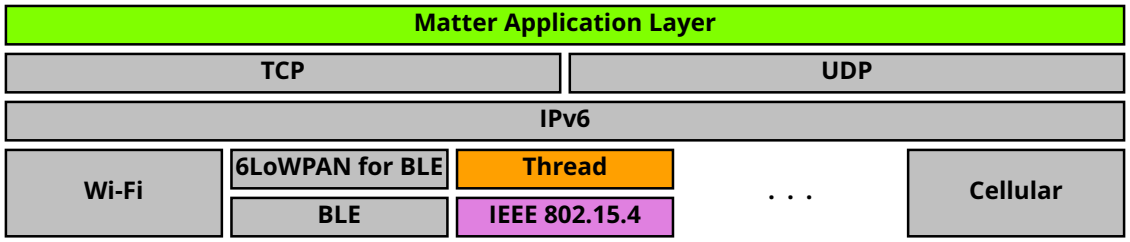


Figure 1: Thread is currently the only networking protocol over IEEE 802.15.4 that is considered for the Matter standard [18].

with development boards that were flashed with OpenThread binaries to operate as Thread devices.

- We demonstrate that an outside attacker can launch a set of energy depletion attacks that prevent an OpenThread-enabled device from returning to its energy-saving sleep mode.
- We present a set of attacks that incorporate selective jamming and spoofing techniques in an attempt to guess the low-entropy password of a Thread device during the commissioning process.
- We are publicly releasing the Thread packets that we captured during our experiments as well as the source code that we wrote in order to study the security of Thread networks.[1]

We reached out to the Thread Group in January 2022 to responsibly disclose our findings and mitigation recommendations, while we continued providing updates about further improvements that we made to our proof-of-concept attacks.

The rest of this paper is organized as follows. In Section 2 we provide relevant background information about the Thread networking protocol, while in Section 3 we review related work on the security of Thread networks. Our threat model and assumptions are provided in Section 4, followed by our discussion on the results of our packet analysis experiments in Section 5. Then, we report the results from our energy depletion attack experiments and our online password guessing attack experiments in Sections 6 and 7 respectively, while we conclude in Section 8.

2 BACKGROUND

In this section we provide a brief overview of the Thread networking protocol [49]. Although Thread Group members have access to the Thread specification and documents [48], there are several publicly accessible sources of information that an independent security analyst could use to learn about the operation of Thread networks.

There is a variety of device types and roles that can be observed in a Thread network [41, 53]. A Thread device can be either a Full Thread Device (FTD) or a Minimal Thread Device (MTD). Unlike FTDs, MTDs are expected to have significant resource constraints, which is reflected in the different roles that they can assume. An FTD can operate as a *Thread Router* in order to provide routing services to other Thread devices, with its receiver being enabled even when it is idle. In each Thread network, one of its Thread Routers is elected to assume the role of the *Thread Leader*, which

[1]Our dataset will be available on CRAWDAD [20], while our source code will be available on the corresponding GitHub repositories.

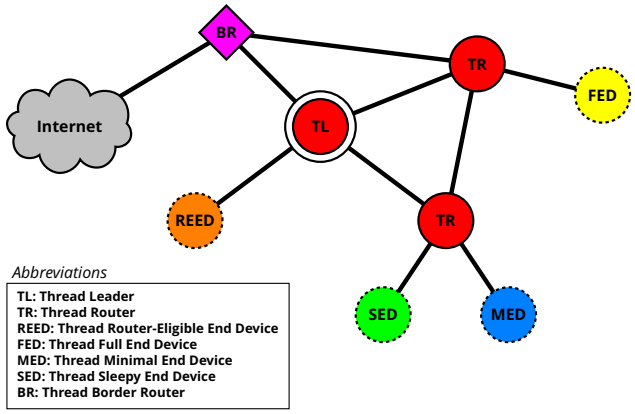


Figure 2: An example of a few roles that can be observed in a Thread network [41, 53].

takes additional responsibilities that are related to the management of the Thread network. If a Thread network becomes partitioned due to lost connectivity, then each partition is treated as a different Thread network that elects its own Thread Leader. If the Thread Leader disconnects at some point, then another Thread Router will be elected to take its place so that the Thread network will continue having a Thread Leader. Note that each Thread network can have up to 32 active Thread Routers. However, an FTD can also operate as a *Thread Router-Eligible End Device*, which does not provide routing services to other Thread devices but can be promoted to a Thread Router through the Thread Leader if the required conditions are met. If an FTD is not capable of operating as a Thread Router, then it operates as a *Thread Full End Device*. A battery-powered MTD can operate as a *Thread Sleepy End Device* in order to conserve its energy by disabling its receiver when it is idle [50]. Alternatively, an MTD could operate as a *Thread Minimal End Device* that keeps its receiver enabled even when it is idle. Two additional roles that an MTD may be able to assume are that of a *Synchronized Sleepy End Device* and that of a *Bluetooth End Device*. The main characteristic of all these End Device roles is that they rely on a Thread Router or the Thread Leader for their routing services. Furthermore, any Thread device that can provide connectivity between a Thread network and a non-Thread network is considered a *Thread Border Router*. In Figure 2 we provide an example of a Thread network topology with some of the roles that we described in this paragraph.

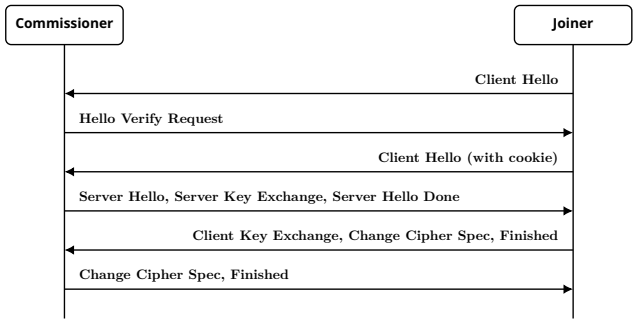


Figure 3: In Thread networks, the commissioner authenticates a joiner through a DTLS handshake, which may have to be relayed by a router [51].

Since Thread networks are IEEE 802.15.4-based networks, each Thread network is using a Personal Area Network Identifier (PAN ID) to distinguish itself from other nearby networks [29, p. 14]. However, each Thread network is also utilizing an Extended PAN ID and a Network Name [40]. Furthermore, note that the IEEE 802.15.4 standard supports the usage of 16-bit short addresses and 64-bit extended addresses on the Medium Access Control (MAC) layer [29, p. 140]. Thread devices are embedding routing information in these addresses that is in turn used for IPv6 addressing purposes [39, 53]. More specifically, Thread devices are transmitting and receiving IPv6 packets by utilizing the 6LoWPAN adaptation layer [27, 35] above the MAC layer [52], where 6LoWPAN stands for IPv6 over Low-Power Wireless Personal Area Network. Two key features of the 6LoWPAN adaptation layer are header compression and packet fragmentation. Thread devices are also using the Mesh Link Establishment (MLE) protocol [31] to configure links and distribute information about the Thread network [53]. It is important to note that MLE frames can be secured using the same security suite as MAC frames (that is, AES-128 [36] in CCM* mode [29, p. 231], which is an extension of the CCM mode [22]), but with a different cryptographic key. We describe how these MAC keys and MLE keys can be derived in the next paragraph.

Thread networks use an elliptic-curve variant of the Password-Authenticated Key Exchange by Juggling (J-PAKE) protocol [25], which incorporates the Schnorr non-interactive zero-knowledge proof mechanism [26], to establish a shared secret between two authenticated devices based on a shared low-entropy password [51]. The authentication is performed during a Datagram Transport Layer Security (DTLS) handshake [45] between an authorized commissioner and a joiner that wants to join the Thread network. Figure 3 shows the messages that are exchanged during a successful DTLS handshake where the commissioner and the joiner communicate directly. If they cannot communicate directly, then the DTLS handshake will have to be relayed by a router. After a successful completion of the Thread commissioning process, the joiner will have received several network parameters including a master key. According to Wireshark[2], given a 128-bit master key K_{Master}, a 32-bit sequence counter c, and an 8-bit key index i, if we first compute a message m as

$$m = ((c \;\&\; \texttt{0x80}) + ((i-1) \;\&\; \texttt{0x7f})) \;||\; \texttt{0x546872656164}, \quad (1)$$

then we can derive the 128-bit MAC key K_{MAC} and the 128-bit MLE key K_{MLE} by computing

$$K_{\text{MLE}} \;||\; K_{\text{MAC}} = \texttt{HMAC\_SHA256}(K_{\text{Master}},\ m), \quad (2)$$

where & denotes the bitwise AND operation, || denotes the concatenation operation, and $\texttt{HMAC\_SHA256}(\cdot,\cdot)$ denotes the keyed-hash message authentication code (HMAC) function [33] that uses the SHA-256 hash function [37]. In words, the MLE key corresponds to the first 128 bits of the output of the HMAC-SHA256 function that has been parameterized by the master key and the computed message, while the last 128 bits correspond to the MAC key. An equivalent derivation process can be observed in OpenThread's implementation of the Thread networking protocol.[3]

3 RELATED WORK

To the best of our knowledge, only a few researchers have analyzed the security of Thread networks in the literature so far. Liu et al. proposed a security assessment taxonomy for building automation systems and applied it to the Thread networking protocol [34]. They identified a number of potential security issues in Thread networks, most of which are inherited from the IEEE 802.15.4 standard. For example, jamming [56] and MAC acknowledgment spoofing [46] are two types of attacks against IEEE 802.15.4-based networks that have been well known for several years. However, note that such attacks can also be combined to launch more sophisticated attacks. Regarding the Thread commissioning process, Liu et al. argued that an attacker could degrade the performance of the network by flooding it with either Beacon Requests or DTLS handshakes. In contrast, the attacks that we present in Section 7 prevent the successful commissioning of legitimate Thread devices that in turn enables the attacker to perform multiple online password guesses in certain scenarios. Dinu and Kizhvatov analyzed an electromagnetic side-channel attack on a development board that was running OpenThread [21]. However, their threat model differs from ours significantly. As we mention in Section 4, physical attacks like the one that Dinu and Kizhvatov analyzed are outside the scope of this work. Instead, we are focusing on attacks that could be launched against nearby Thread devices over the communication channel, without having prior knowledge of any cryptographic keys.

Given that both Zigbee and Thread are based on the IEEE 802.15.4 standard, we wanted to test whether battery-powered Thread devices could be susceptible to a known energy depletion attack against battery-powered Zigbee devices, even though Thread networks utilize MAC-layer security services that are disabled in Zigbee networks. Cao et al. proposed an energy depletion attack, where an outside attacker is transmitting spoofed packets with supposedly encrypted and authenticated data to trick a targeted Zigbee node into wasting its energy by receiving those packets and performing unnecessary security computations [14]. Akestoridis and Tague improved upon the aforementioned attack by selectively jamming Data Requests and making the aggressiveness of the energy depletion attack configurable [7]. However, the existing implementation

[2] https://gitlab.com/wireshark/wireshark/-/blob/5ecb57cb9026cebf0cfa4918c4a86942620c5ecf/epan/dissectors/packet-thread.c

[3] https://github.com/openthread/openthread/blob/395d502576025f432e37da5538abf53ed4615700/src/core/thread/key_manager.cpp

of this energy depletion attack cannot be launched against a Thread Sleepy End Device because of differences in the formatting of Zigbee packets and Thread packets. As we explain in Section 6, we were able to launch similar energy depletion attacks against our OpenThread-enabled device by modifying the condition for the selective jamming of Data Requests and the format of the spoofed packets. More specifically, we adapted the jamming condition to match the formatting of Data Requests that are transmitted by Thread Sleepy End Devices, while we also injected five different spoofed packet types in order to test whether our Thread Sleepy End Device would waste its energy by processing them over long periods of time instead of conserving its energy in sleep mode.

4 THREAT MODEL AND ASSUMPTIONS

We consider an attacker that has no prior knowledge of any cryptographic keys for our threat model. The security objectives that we have set for Thread networks are (a) authenticity, (b) integrity, (c) confidentiality, and (d) availability. The attacker attempts to violate as many security objectives as possible, with their primary goal being to obtain the cryptographic keys of a targeted Thread network. We assume that the attacker has the required hardware and software tools within communication range of the targeted Thread network in order to (a) capture, (b) transmit, (c) jam, and (d) analyze Thread packets. Regarding the end user and their devices, we assume that they are not deliberately downgrading the security of the Thread network, with physical attacks being outside the scope of this work. Furthermore, regarding the security of the Thread commissioning process, we are focusing on the scenario where the commissioner communicates with the joiner directly.

If the attacker succeeds in obtaining the cryptographic keys of a targeted Thread network, then they could launch several impersonation attacks against it. Such a scenario would be especially concerning if the application layer above Thread relies on its cryptographic keys as well, since that would enable the attacker to launch command injection attacks that would change the state of actuators and to decrypt potentially sensitive sensor data. However, even if the application layer above Thread utilizes security services with a different set of cryptographic keys, an attacker that has obtained the master key of a Thread network could still launch several disruptive network-layer attacks such as spoofing routing information and dropping packets that it should forward [30, 55].

5 PACKET ANALYSIS EXPERIMENTS

Reconnaissance is typically considered the first phase of an intrusion into a traditional computer network [13, 28]. Similarly, an outside attacker would typically start by launching reconnaissance attacks in order to gather information about the Thread devices that they would like to target. Since the attacker does not have any prior knowledge about the end user's cryptographic keys, they have to rely on information that can be inferred either directly or indirectly from unencrypted data. In Section 5.1 we describe the experiments that we conducted in order to study what information an outside attacker could infer from a nearby Thread network. We delineate our observations in Section 5.2, which enabled us to develop the energy depletion attacks and the online password guessing attacks that we present in Sections 6 and 7 respectively.

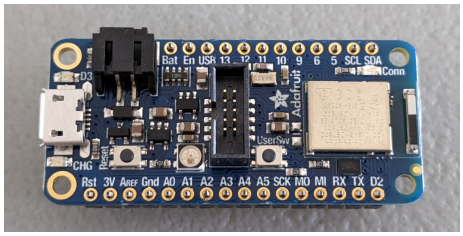


Figure 4: One of the Adafruit Feather nRF52840 Express devices [1] that we used in order to form Thread networks, which were flashed with OpenThread binaries for Nordic Semiconductor nRF528xx SoCs [42].

5.1 Setup

We used four Adafruit Feather nRF52840 Express devices [1], one of which is shown in Figure 4, in order to conduct our experiments. We flashed these devices with OpenThread binaries for Nordic Semiconductor nRF528xx SoCs [42] to make them operate as Thread devices that we configured as follows:

- One device was flashed with an FTD OpenThread binary that had the `COMMISSIONER` and `JOINER` functionalities enabled. This device was used as a legitimate commissioner that would initially operate as a Thread Leader.
- One device was flashed with an FTD OpenThread binary that had only the `JOINER` functionality enabled. This device was used as a legitimate joiner that would operate as a Thread Router shortly after joining a Thread network.
- One device was flashed with an MTD OpenThread binary that had only the `JOINER` functionality enabled. This device was used as a legitimate joiner that would operate as a Thread Sleepy End Device shortly after joining a Thread network.
- One device was flashed with an FTD OpenThread binary that had the `COMMISSIONER` and `JOINER` functionalities enabled. This device was used for impersonation purposes.

We conducted 19 experiments in order to study a representative sample of Thread packets, which included (a) observing Thread traffic during idle periods, (b) commissioning with and without specifying the joiner's 64-bit IEEE address, (c) causing commissioning errors, (d) generating ping and UDP messages, (e) disconnecting and reconnecting a Thread device, (f) downgrading a Thread Router, and (g) attempting to cause PAN ID conflicts. We used a USRP N210 [23] and GNU Radio [24] in order to capture the packets that our Thread devices transmitted. In particular, we generated 19 pcap files by utilizing the `gr-foo` and `gr-ieee802-15-4` modules [9–11], along with the IEEE 802.15.4 transceiver flow graph of the `grc-ieee802154` repository [2]. We used Wireshark [54] for the manual inspection of our pcap files, with a configuration profile that we developed for Thread traffic.[4] Furthermore, we enhanced the packet dissection capabilities of Scapy [47] in order to inspect our pcap files programmatically.[5] Our Scapy enhancements include improvements to the dissection of beacons, secured MAC frames, and 6LoWPAN header fields, as well as the creation of an `mle` layer

[4] https://github.com/akestoridis/wireshark-thread-profile
[5] Our enhancements have been submitted to Scapy's repository as a pull request.

for the identification of different MLE commands. In addition, we enhanced the security analysis capabilities of Zigator [4] to support Thread networks.[6] We used our enhanced version of Zigator to store the header fields of our captured Thread packets in an SQLite database for the subsequent execution of SQL queries, as well as to inject forged packets with an ATUSB [44] during the experiment where we attempted to cause PAN ID conflicts.

5.2 Observations

5.2.1 MAC Frame Types. According to the IEEE 802.15.4-2006 standard, there are four possible MAC frame types [29, p. 139]: beacons, MAC Data packets, MAC acknowledgments, and MAC commands. Although we observed all four of these MAC frame types during our experiments, we only observed two out of the nine possible MAC commands [29, p. 149], namely: Data Requests and Beacon Requests. In contrast, Zigbee networks have been observed to utilize six MAC commands [5]. For example, while Zigbee devices rely on Association Requests and Association Responses to join a Zigbee network, Thread devices perform the DTLS handshake over MAC Data packets to join a Thread network. Note that even though all Data Requests that we captured during our experiments were secured by 32-bit Message Integrity Codes, this does not preclude their selective jamming because they are the only MAC commands that are transmitted by Thread devices with a length of either 22 or 34 bytes (depending on the packet's addressing mode). Interestingly enough, even if other MAC commands were utilized with the same packet lengths, according to the IEEE 802.15.4-2006 standard, the Command Identifier field of MAC commands is transmitted unencrypted even when MAC-layer security is enabled [29, p. 200]. Thus, it is possible for an outside attacker to identify Data Requests, as they are being transmitted by Thread devices, and selectively jam them in order to prevent their successful delivery. We take advantage of this observation to develop the energy depletion attacks that we describe in Section 6.

5.2.2 MLE Command Types. The format of MLE frames has been defined in an Internet-Draft, along with seven MLE command types [31]. However, Wireshark is currently recognizing 18 MLE command types.[7] We observed 14 out of these 18 MLE commands during our experiments, which are shown in Table 1 along with the MAC-layer and MLE-layer security services that they utilized, most of which utilized security services only on the MLE layer. For example, the payload of each Advertisement command that we captured was encrypted and appended with a 32-bit Message Integrity Code. Interestingly enough, unlike secured MAC commands, MLE commands that utilized MLE-layer confidentiality services were encrypting their Command Type fields along with their payloads. The only MLE commands that we observed with both MAC and MLE security enabled were Announce commands. Notably, Discovery Requests and Discovery Responses did not utilize any security services since they are transmitted before a joiner initiates a DTLS handshake. We further discuss how an attacker can leverage this fact for impersonation purposes in Section 7.

[6] Our enhancements have been pushed to Zigator's repository.
[7] https://gitlab.com/wireshark/wireshark/-/blob/5ecb57cb9026cebf0cfa4918c4a86942620c5ecf/epan/dissectors/packet-mle.c

Table 1: The MLE command types that we observed during our experiments, along with the MAC-layer and MLE-layer security attributes that they used, if enabled.

MLE Command Type	MAC Security	MLE Security
Link Request	Disabled	ENC-MIC-32
Link Accept	Disabled	ENC-MIC-32
Link Accept and Request	Disabled	ENC-MIC-32
Advertisement	Disabled	ENC-MIC-32
Data Response	Disabled	ENC-MIC-32
Parent Request	Disabled	ENC-MIC-32
Parent Response	Disabled	ENC-MIC-32
Child ID Request	Disabled	ENC-MIC-32
Child ID Response	Disabled	ENC-MIC-32
Child Update Request	Disabled	ENC-MIC-32
Child Update Response	Disabled	ENC-MIC-32
Announce	ENC-MIC-32	ENC-MIC-32
Discovery Request	Disabled	Disabled
Discovery Response	Disabled	Disabled

5.2.3 Unencrypted Header Fields. In addition to most of our captured MLE commands, we also observed other MAC Data packets that were transmitted without utilizing MAC-layer security services. These corresponded to fragmented and unfragmented 6LoWPAN packets that were used to perform the DTLS handshake of the Thread commissioning process. All of these unsecured MAC Data packets are transmitting valuable information unencrypted that an outside attacker can incorporate into their attacks, such as addressing fields and port numbers. However, it should be noted that even with MAC-layer security enabled, the MAC-layer header fields are still being transmitted unencrypted. Furthermore, similar to Discovery Requests and Discovery Responses, all the header fields of Beacon Requests and Thread beacons were transmitted unencrypted. Lastly, we observed several packet types that were relying on unsecured MAC acknowledgments, including Data Requests and 6LoWPAN fragments, which we took into account as we were developing the attacks that we describe in Sections 6 and 7.

5.2.4 PAN ID Conflict Attempts. In the past, PAN ID conflict attacks have been used against Zigbee devices in order to disconnect them from their networks [5, 12]. In order to test whether our Thread network is susceptible to a similar denial-of-service attack, we attempted to cause PAN ID conflicts during an experiment by injecting Zigbee and Thread beacons that were using the same PAN ID as our Thread network, as well as setting another Thread Leader to form a different Thread network with the same PAN ID as our original Thread network while we were injecting Beacon Requests. We did not observe any reaction from our original Thread network when we performed the aforementioned PAN ID conflict attempts. We incorporate this observation into one of the online password guessing attacks that we present in Section 7.

6 ENERGY DEPLETION ATTACKS

Similar to the case of a battery-powered Zigbee device [7], there are multiple reasons that could motivate an outside attacker to launch

an energy depletion attack against a battery-powered Thread device. For example, the attacker could target a specific battery-powered Thread device that they would like to prevent from sending and receiving packets until its battery is replaced. The attacker could also launch such an attack in order to force the end user to replace the batteries of their Thread devices more frequently than typically expected and either increase the maintenance cost of their Thread devices or to simply abandon them. Furthermore, by depleting the energy of a battery-powered Thread device, the attacker could trick the end user into factory resetting it and recommissioning it to their Thread network, as we further discuss in Section 7. In Section 6.1 we delineate a set of energy depletion attacks that we hypothesized, while in Section 6.2 we describe the setup of the experiments that we conducted in order to test them. We report the results of our energy depletion attack experiments in Section 6.3 and we provide mitigation recommendations in Section 6.4.

6.1 Hypotheses

6.1.1 Spoofing Secured MAC Data Packets. As we explained in Section 5.2.1, an outside attacker can selectively jam the Data Requests that Thread devices transmit to poll for pending packets. The first spoofed packet type that we decided to test injecting, after the selective jamming of a Data Request and the spoofing of a MAC acknowledgment with the Frame Pending field set to one, was that of a supposedly secured MAC Data packet. More specifically, the attacker could use the short addresses from previous Data Requests of the targeted Thread Sleepy End Device to forge them with the Frame Pending field always set to one in order to trick it into wasting its energy instead of returning to its energy-saving sleep mode. However, given that the Frame Pending field is a MAC header field and that the verification process would expectedly fail on the MAC layer for such packets, we decided to test additional spoofed packet types in case that approach turned out to be ineffective.

6.1.2 Spoofing Secured MLE Commands. Based on our observations in Section 5.2.2 about the security services that legitimate MLE commands were utilizing, we decided to also test the injection of MLE commands that are supposedly secured only on the MLE layer. The attacker could forge them using the extended addresses from legitimate MLE commands, as well as 34-byte Data Requests, but with the Frame Pending field always set to one. The rationale behind this decision was that there may be a lack of cross-layer communication regarding the Frame Pending field on the MAC layer and the result of the verification process on the MLE layer.

6.1.3 Spoofing Invalid MLE Commands. Given that the MLE layer uses the same security suite as the MAC layer [31], we considered the scenario where the result of the verification process could affect the transmission of new Data Requests regardless of whether it was performed on the MAC or the MLE layer. For that reason, we decided to test injecting supposedly secured MLE commands with deliberately incorrect UDP checksums because, after the decompression of their IPv6 and UDP headers, they could be discarded without MLE-layer verification and without informing the component that is responsible for the transmission of new Data Requests.

6.1.4 Spoofing Unsecured 6LoWPAN Fragments. The last spoofed packet type that we decided to test injecting was that of an unsecured 6LoWPAN fragment. Unlike the aforementioned spoofed packet types, the receiver of an unsecured 6LoWPAN fragment may not be able to verify the authenticity of the message until they reassemble all of its fragments. The attacker could then keep transmitting fragments of supposedly different messages by using different datagram tag values. Depending on how the receiver manages the storage of unassembled fragments, the attacker could potentially trick the receiver into exhausting their memory with such an attack. Although we did not observe any fragmented MLE commands during our packet analysis experiments, we decided to use the source and destination port numbers that MLE commands use for our spoofed first fragments. Furthermore, it is important to note that a receiver may process a subsequent fragment that arrived before the first fragment [35]. For that reason, we decided to test the injection of first fragments and the injection of subsequent fragments separately because subsequent fragments of UDP messages do not have to include source and destination port numbers.

6.2 Setup

We used the same packet capturing tools for our energy depletion attack experiments, while forming Thread networks that consisted of only a Thread Leader and a Thread Sleepy End Device, that we described in Section 5.1. More specifically, we conducted five energy depletion attack experiments that differed only in terms of the type of the spoofed packets, which corresponded to (a) secured MAC Data packets, (b) secured MLE commands with correct UDP checksums, (c) secured MLE commands with incorrect UDP checksums, (d) unsecured 6LoWPAN first fragments, and (e) unsecured 6LoWPAN subsequent fragments. We modified the existing implementation of an energy depletion attack against battery-powered Zigbee devices of the `atusb-attacks` repository [3] in order to launch our energy depletion attacks with an ATUSB [44] against our Thread Sleepy End Device.[8] Instead, we considered an energy depletion attack experiment successful if our ATUSB could cause our Thread Sleepy End Device to keep transmitting new Data Requests over a long period of time, similar to how commercial battery-powered Zigbee devices have behaved during this type of attack [6]. We configured our ATUSB to selectively jam 22-byte MAC commands of our Thread network and inject spoofed packets for 30 seconds before allowing our Thread devices to communicate normally for 3 seconds and then wait for the next Data Request to repeat the same process. Furthermore, after the completion of the aforementioned experiments, we used a USB tester to collect power measurements of our Thread Sleepy End Device during our energy depletion attack where supposedly secured MLE commands were injected.

6.3 Findings

As we can see in Figure 5a and Figure 5b, our ATUSB was able to selectively jam each 22-byte Data Request that our Thread Sleepy End Device transmitted, as indicated by the 1-byte and 22-byte invalid packets that we captured, and then spoof a MAC acknowledgment with the expected MAC sequence number. After that, our ATUSB

[8] Our proof-of-concept attacks will be available on the `atusb-attacks` repository.

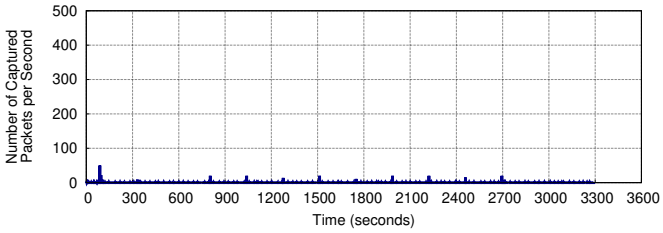
No.	Time	Delta time	MAC SN	Length	Info
141	794.851685	28.334326	168	69	Advertisement
142	803.071885	8.220200		1	[Malformed Packet]
143	803.072786	0.000901	69	5	Ack
144	803.077326	0.004540	255	127	Data, Dst: 0x3c01, Src: 0x3
145	803.077867	0.000541	255	5	Ack
146	803.175519	0.097652		1	[Malformed Packet]
147	803.176426	0.000907	70	5	Ack
148	803.180968	0.004542	255	127	Data, Dst: 0x3c01, Src: 0x3
149	803.181506	0.000538	255	5	Ack
150	803.277213	0.095707		1	[Malformed Packet]
151	803.278111	0.000898	71	5	Ack
152	803.282657	0.004546	255	127	Data, Dst: 0x3c01, Src: 0x3
153	803.283194	0.000537	255	5	Ack
154	803.379560	0.096366		1	[Malformed Packet]
155	803.380447	0.000887	72	5	Ack
156	803.384970	0.004523	255	127	Data, Dst: 0x3c01, Src: 0x3
157	803.385514	0.000544	255	5	Ack
158	803.481527	0.096013		1	[Malformed Packet]
159	803.482433	0.000906	73	5	Ack
160	803.487002	0.004569	255	127	Data, Dst: 0x3c01, Src: 0x3
161	803.487513	0.000511	255	5	Ack
162	823.025983	19.538470	169	69	Advertisement
163	861.188325	38.162342	170	69	Advertisement
164	892.261120	31.072795	171	69	Advertisement
165	919.325493	27.064373	172	69	Advertisement
166	956.190388	36.864895	173	69	Advertisement
167	978.504926	22.314538	174	69	Advertisement
168	1023.724425	45.219499	175	69	Advertisement
169	1039.483289	15.758864		1	[Malformed Packet]
170	1039.484195	0.000906	74	5	Ack
171	1039.488728	0.004533	255	127	Data, Dst: 0x3c01, Src: 0x3
172	1039.489273	0.000545	255	5	Ack
173	1039.585382	0.096109		1	[Malformed Packet]
174	1039.586288	0.000906	75	5	Ack
175	1039.590826	0.004538	255	127	Data, Dst: 0x3c01, Src: 0x3

(a)

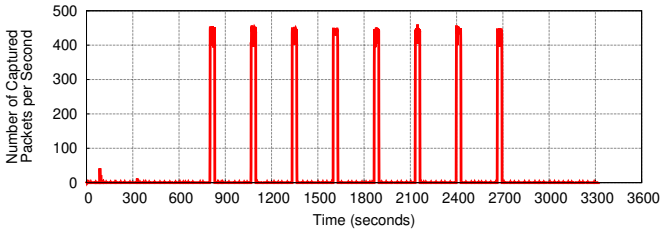
No.	Time	Delta time	MAC SN	Length	Info
144	799.074460	44.457671	241	69	Advertisement
145	800.781643	1.707183		1	[Malformed Packet]
146	800.782554	0.000911	134	5	Ack
147	800.787540	0.004986	255	127	
148	800.788082	0.000542	255	5	Ack
149	800.789621	0.001539		1	[Malformed Packet]
150	800.790529	0.000908	135	5	Ack
151	800.795523	0.004994	255	127	
152	800.796070	0.000547	255	5	Ack
153	800.798518	0.002448		1	[Malformed Packet]
154	800.799427	0.000909	136	5	Ack
155	800.804415	0.004988	255	127	
156	800.804965	0.000550	255	5	Ack
157	800.807773	0.002808		1	[Malformed Packet]
158	800.808680	0.000907	137	5	Ack
159	800.813670	0.004990	255	127	
160	800.814214	0.000544	255	5	Ack
161	800.818576	0.004362	138	5	Ack
162	800.823561	0.004985	255	127	
163	800.824105	0.000544	255	5	Ack
164	800.827738	0.003633	139	22	Unknown Command, Bad FCS
165	800.828468	0.000730	139	5	Ack
166	800.833454	0.004986	255	127	
167	800.833998	0.000544	255	5	Ack
168	800.835813	0.001815		1	[Malformed Packet]
169	800.836725	0.000912	140	5	Ack
170	800.841715	0.004990	255	127	
171	800.842259	0.000544	255	5	Ack
172	800.845706	0.003447		1	[Malformed Packet]
173	800.846619	0.000913	141	5	Ack
174	800.851603	0.004984	255	127	
175	800.852148	0.000545	255	5	Ack
176	800.853604	0.001456		1	[Malformed Packet]
177	800.854511	0.000907	142	5	Ack
178	800.859591	0.005080	255	127	

(b)

Figure 5: Screenshots of captured packets during energy depletion attack experiments, where the attacker was spoofing either (a) a secured MAC Data packet or (b) a secured MLE command, after the selective jamming of a Data Request and the spoofing of a corresponding MAC acknowledgment.



(a)



(b)

Figure 6: Number of captured packets per second during energy depletion attack experiments, where the attacker was spoofing either (a) a secured MAC Data packet or (b) a secured MLE command, after the selective jamming of a Data Request and the spoofing of a corresponding MAC acknowledgment.

injected a spoofed packet that our Thread Sleepy End Device acknowledged. In the case where a supposedly secured MAC Data packet was injected after each spoofed MAC acknowledgment, our Thread Sleepy End Device transmitted a new Data Request only five times and then it appears that it returned to its sleep mode, with the same pattern repeating during its next polling period. Therefore, the impact of this attack is expected to be low because the targeted Thread Sleepy End Device would still spend most of its time in sleep mode. This is also evident from Figure 6a, which shows only a small increase in the number of captured packets per second during this attack. However, when we were injecting any of the other spoofed packet types that we tested, our Thread Sleepy End Device kept transmitting new Data Requests until our ATUSB allowed one of them to reach our Thread Leader as we next discuss.

The attack where secured MLE commands were injected (as shown in Figure 5b) is expected to have the highest impact among the energy depletion attacks that we tested because the targeted Thread Sleepy End Device would keep performing unnecessary security computations as part of the verification process for all these spoofed packets. As we can observe in Figure 6b, the number of captured packets per second increased significantly during each 30-second period that our ATUSB was selectively jamming Data Requests and injecting supposedly secured MLE commands. Furthermore, according to the power measurements that we collected with our USB tester and provide in Figure 7, the power of our Thread

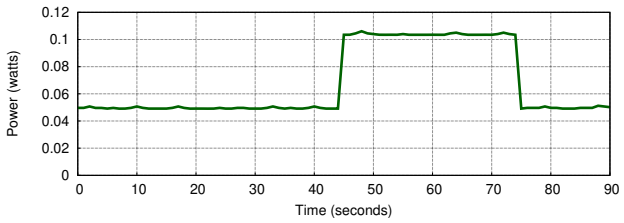


Figure 7: Collected power measurements of our development board that operated as a Thread Sleepy End Device, from about 45 seconds before the selective jamming of its Data Requests and injection of spoofed MAC acknowledgments and supposedly secured MLE commands.

Sleepy End Device was approximately 0.05 watts shortly before our attack started, while it was approximately 0.104 watts during the spoofing of secured MLE commands by our ATUSB. Regarding the attack where invalid MLE commands were injected, even though the targeted Thread Sleepy End Device may not perform any unnecessary security computations in this case, the impact of this attack is still expected to be significant because it would be wasting its energy receiving and transmitting packets for long periods of time instead of conserving its energy in sleep mode. The same argument applies for the injection of unsecured 6LoWPAN first fragments and unsecured 6LoWPAN subsequent fragments. Regarding the receiver's potential memory exhaustion that we hypothesized in Section 6.1.4, we did not observe any immediate interruptions in its operation. Lastly, we did not observe any difference in the handling of first and subsequent fragments, making both of them potential options for energy depletion purposes.

6.4 Recommendations

While there is a straightforward mitigation strategy for the second and third energy depletion attack (that is, making the result of the MLE verification process and the UDP integrity check affect the transmission of new Data Requests), mitigating the fourth energy depletion attack is not as simple because, as we mentioned in Section 5.2.3, unsecured 6LoWPAN fragments are being legitimately used in Thread networks. One potential approach is to make Thread Sleepy End Devices ignore unsecured 6LoWPAN fragments that are destined for ports where unsecured fragmentation is not expected. However, it is important to keep in mind that, according to RFC 4944, a receiver may process a subsequent fragment that arrived before the first fragment [35]. This complicates the development of an effective mitigation strategy because a subsequent fragment may not include a destination port number.

7 ONLINE PASSWORD GUESSING ATTACKS

As noted in RFC 8236, there is nothing to prevent an online attacker from trying a random guess of the shared low-entropy password during the key exchange process [25]. Furthermore, it appears that RFC 8236 only provides recommendations for the handling of online password guessing attacks. This motivated us to implement proof-of-concept attacks in order to study the feasibility and associated risk of such attacks against OpenThread-enabled devices. We present our online password guessing attacks in Section 7.1 and we describe the setup of the corresponding experiments that we conducted in Section 7.2. In Section 7.3 we discuss the results of these experiments, while in Section 7.4 we provide recommendations regarding the security of the Thread commissioning process.

7.1 Hypotheses

7.1.1 Impersonating the Joiner. If the OpenThread user does not specify the 64-bit IEEE address of the Thread device that they want to commission, then the attacker could perform multiple password guesses during the period that the commissioner is accepting new Thread devices. Furthermore, the attacker could potentially trick the end user into restarting the commissioning process by preventing the legitimate Thread device from joining their Thread network to perform more password guesses. The attacker could achieve that by selectively jamming unsecured first fragments that use the joiner UDP port, which would prevent the reassembly of Client Hello messages. More specifically, in order to allow only the impersonating joiner to complete the DTLS handshake, the selective jammer should ignore first fragments that are exchanged between two specified extended MAC addresses (the commissioner's and the impersonator's). In addition, since 6LoWPAN fragments rely on unsecured MAC acknowledgments, the attacker could spoof one after each selectively jammed first fragment to trick the corresponding Thread device into thinking that it was received successfully.

7.1.2 Impersonating the Commissioner. Alternatively, instead of impersonating a joiner to perform online password guesses, an attacker could also impersonate a commissioner. Note that, in order for a joiner to attempt to join a Thread network for the first time, they first have to learn some basic information about the end user's Thread network (such as its PAN ID), which they can request by transmitting Beacon Requests and Discovery Requests. However, an attacker could set up a device to operate as a fully functional commissioner for a different Thread network that would accept a guessed password, while another device is selectively jamming the Thread beacons and Discovery Responses of the legitimate Thread network in order to trick the joiner into initiating a DTLS handshake with the impersonating commissioner. Even if the legitimate commissioner specified the joiner's 64-bit IEEE address, the impersonating commissioner can simply be configured to accept any joiner. Furthermore, even if the joiner was already aware of the legitimate commissioner's PAN ID, the impersonating commissioner could also be configured to use the same PAN ID without causing a PAN ID conflict, as we discovered during the experiment that we discussed in Section 5.2.4. Therefore, in order to trick the joiner into interacting with the impersonating commissioner instead of the legitimate commissioner, the attacker can selectively jam Thread beacons and Discovery Responses with the corresponding PAN ID, except for the ones whose source address corresponds to the extended address of the impersonating commissioner.

7.2 Setup

We conducted 11 experiments in order to study the feasibility and associated risk of the online password guessing attacks that we hypothesized, while using the packet capturing tools that we described in Section 5.1. We used FTD joiners for all of our online password

No.	Time	Src IPv6 address	Src port	Dst IPv6 address	Dst port	MAC SN	Length	Info
19	207.714883					0	10	Beacon Request
20	207.719195					62	45	Beacon, Src: 0x0000, Ne
21	213.236549	fe80::58a6:96bb:4ce7:79a1	19788	ff02::2	19788	156	37	Discovery Request
22	213.263176	fe80::146c:96da:d900:b556	19788	fe80::58a6:96bb:4ce7:79a1	19788	117	76	Discovery Response
23	213.263719					117	5	Ack
24	214.159250	fe80::d84e:c7c4:cdfa:9dea	19788	ff02::2	19788	39	37	Discovery Request
25	214.242875	fe80::146c:96da:d900:b556	19788	fe80::d84e:c7c4:cdfa:9dea	19788	118	76	Discovery Response
26	214.243363					118	5	Ack
27	216.029554						72	Reserved
28	216.037940					165	5	Ack
29	216.044976					166	124	Data, Dst: 16:6c:96:da:
30	216.045381					166	5	Ack
31	216.052003					167	124	Data, Dst: 16:6c:96:da:
32	216.052596					167	5	Ack
33	216.057635					168	124	Data, Dst: 16:6c:96:da:
34	216.058176					168	5	Ack
35	216.061989					169	87	Data, Dst: 16:6c:96:da:
36	216.062531					169	5	Ack
37	216.955793					48	124	Data, Dst: 16:6c:96:da:
38	216.956325					48	5	Ack
39	216.962591					49	124	Data, Dst: 16:6c:96:da:
40	216.963129					49	5	Ack
41	216.968855					50	124	Data, Dst: 16:6c:96:da:
42	216.969396					50	5	Ack
43	216.974114					51	124	Data, Dst: 16:6c:96:da:
44	216.974658					51	5	Ack
45	216.979384	fe80::dc27:8dc6:725f:ac0d	1000	fe80::146c:96da:d900:b556	1000	52	87	Client Hello
46	216.979919					52	5	Ack
47	217.099714	fe80::146c:96da:d900:b556	1000	fe80::dc27:8dc6:725f:ac0d	1000	120	92	Hello Verify Request
48	217.100252					120	5	Ack

Figure 8: Screenshot of captured packets during an online password guessing attack experiment, where the attacker selectively jammed the unsecured 6LoWPAN first fragments of the legitimate joiner, while spoofing a MAC acknowledgment for each selectively jammed packet, in order to prevent the reassembly of the legitimate joiner's Client Hello message and to perform online password guesses by impersonating a joiner.

guessing attack experiments, which interacted with a commissioner directly. We implemented the selective jamming functionality of our attacks[9] by utilizing the framework of the `atusb-attacks` repository [3], which we then launched with an ATUSB [44].

7.3 Findings

As we can see in Figure 8, our ATUSB was able to selectively jam the first fragment of the legitimate joiner's Client Hello message (indicated by the 72-byte invalid packet that was captured) and to spoof a corresponding MAC acknowledgment. This caused the legitimate joiner to continue transmitting the remaining fragments of the Client Hello message, after which the legitimate joiner was anticipating to receive a Hello Verify Request message. However, the commissioner was still waiting to receive the first fragment of the legitimate joiner's Client Hello message. After a few seconds, the legitimate joiner attempted to transmit a new Client Hello message, with the same pattern repeating for about 2 minutes (the default duration of a commissioning period). During a single commissioning period, we were able to test 13 deliberately incorrect passwords by using OpenThread's CLI with an impersonating joiner, whose first fragments were not selectively jammed (as indicated by the successful reassembly of the Client Hello message and the corresponding Hello Verify Request message in Figure 8). While this is a small number of password guesses given the much larger number of possible passwords, in a real-world setting, an attacker could trick the end user into restarting the commissioning process by preventing a legitimate Thread device from joining the end user's Thread network to perform more password guesses. In addition, if the password is hard coded, the attacker could try guessing more passwords by tricking the end user into recommissioning their device by depleting their energy or by using additional jamming techniques, potentially similar to those that have been suggested against legacy Zigbee devices [58] and Zigbee 3.0 devices [5]. However, given that OpenThread imposes a minimum password length of six characters from an alphabet of 32 possible characters,[10] the risk of such an attack should be low as long as the end user does not accept new Thread devices for long periods of time and does not use an easily guessable password.

In Figure 9 we provide a screenshot of captured packets during the experiment where our ATUSB was selectively jamming the Thread beacons and Discovery Responses of our legitimate Thread network, which caused our legitimate joiner to initiate a DTLS handshake with our impersonating commissioner that was accepting new Thread devices with an incorrectly guessed password. At the time of writing, even if OpenThread users are aware of the PAN ID of their Thread networks, they cannot provide that information to a joiner in order to avoid initiating a DTLS handshake with another Thread network.[11] Nevertheless, OpenThread users could create a new operational dataset with the corresponding PAN ID[12] before executing the `joiner start` command, even though this does not appear to be a typical step for the commissioning of OpenThread-enabled devices [38]. This caused the Discovery Requests of the legitimate joiner to use the specified PAN ID as their destination PAN ID (instead of using the broadcast PAN ID `0xffff`). However, by configuring the impersonating commissioner to also

[9] Our proof-of-concept attacks will be available on the `atusb-attacks` repository.

[10] https://github.com/openthread/openthread/blob/395d502576025f432e37da5538abf53ed4615700/src/core/meshcop/meshcop.cpp

[11] https://github.com/openthread/openthread/blob/395d502576025f432e37da5538abf53ed4615700/src/cli/README_JOINER.md

[12] https://github.com/openthread/openthread/blob/395d502576025f432e37da5538abf53ed4615700/src/cli/README_DATASET.md

No.	Time	Src IPv6 address	Src port	Dst IPv6 address	Dst port	MAC SN	Length	Info
36	302.295287					0	10	Beacon Request
37	302.298065					31	45	Beacon, Src: 0x0000, Ne
38	302.300500					39	25	Beacon
39	303.361149	fe80::84ce:373c:63a8:47d4	19788	ff02::1	19788	152	69	Advertisement
40	312.407310	fe80::a4ef:20b9:fb34:4684	19788	ff02::1	19788	87	69	Advertisement
41	313.411130	fe80::8420:9be4:a397:62e0	19788	ff02::2	19788	46	37	Discovery Request
42	313.509595					88	91	Data, Dst: 86:20:9b:e4:
43	313.514758					144	37	Reserved
44	313.520802					144	37	Reserved
45	313.526234					144	37	Reserved
46	313.530823					144	37	Reserved
47	313.535360					88	91	Data, Dst: 86:20:9b:e4:
48	313.545891					144	37	Reserved
49	313.550245					144	37	Reserved
50	313.555959					144	37	Reserved
51	313.561955					144	37	Reserved
52	313.566305					144	37	Reserved
53	313.571659					144	37	Reserved
54	313.576795					144	37	Reserved
55	313.582406					144	37	Reserved
56	313.586450					144	37	Reserved
57	313.590898					144	37	Reserved
58	313.666585	fe80::84ce:373c:63a8:47d4	19788	fe80::8420:9be4:a397:62e0	19788	153	76	Discovery Response
59	313.667131					153	5	Ack
60	316.208494					55	124	Data, Dst: 86:ce:37:3c:
61	316.209028					55	5	Ack
62	316.215021					56	124	Data, Dst: 86:ce:37:3c:
63	316.215562					56	5	Ack
64	316.222192					57	124	Data, Dst: 86:ce:37:3c:
65	316.222732					57	5	Ack
66	316.228087					58	124	Data, Dst: 86:ce:37:3c:
67	316.228631					58	5	Ack
68	316.233992	fe80::2ccb:8062:7fc5:58de	1000	fe80::84ce:373c:63a8:47d4	1000	59	87	Client Hello
69	316.234535					59	5	Ack
70	316.346250	fe80::84ce:373c:63a8:47d4	1000	fe80::2ccb:8062:7fc5:58de	1000	154	92	Hello Verify Request
71	316.346781					154	5	Ack

Figure 9: Screenshot of captured packets during an online password guessing attack experiment, where the attacker selectively jammed the Thread beacons and Discovery Responses of the legitimate Thread network in order to trick the joiner into initiating a DTLS handshake with the impersonating commissioner.

use the same PAN ID, the behavior that we observed was similar to the one shown in Figure 9. Note that if the attacker already knows the joiner's password, then this attack could also be used in order to hijack the joiner during its commissioning process. This could be concerning if the corresponding password is hard coded, similar to how Zigbee 3.0 devices are currently using hard-coded install codes [57, p. 73], since in that case it could be leaked if it is not properly secured.

7.4 Recommendations

As noted in RFC 8236, consecutively incorrect password guesses can be easily detected with a false authentication counter in order to thwart subsequent password guesses [25]. In real-world deployments, a warning should be raised whenever multiple incorrect password guesses are detected so that the end user can stop the commissioning process and take appropriate actions. Furthermore, we recommend that commercial Thread devices are manufactured to provide a dedicated indication whenever the commissioning process fails because the commissioner expected a different password, so that the end users would not restart the commissioning process and enable the attacker to perform more password guesses. Similarly, we recommend raising a warning whenever a PAN ID conflict is observed so that the end user can prevent the attacker from interacting with the legitimate joiner. Finally, to enhance the security of the commissioning process, commercial Thread devices should ideally be manufactured to enable end users to change their passwords over an out-of-band communication channel, so that the end users are not forced to use hard-coded secrets in cases where they cannot be certain that they have not been leaked.

8 CONCLUSION

In this work we present the results of our analysis on the security of Thread networks that we formed using development boards that were flashed with OpenThread binaries. In order to conduct our experiments, we repurposed hardware and software tools that have been used to analyze the security of Zigbee networks. Since both Zigbee and Thread are based on the IEEE 802.15.4 standard, we were able to enhance the capabilities of existing software tools to support the security analysis of Thread networks. We made several observations about the nature of Thread traffic, which led to the development of energy depletion attacks and online password guessing attacks. Furthermore, we study the susceptibility of our Thread networks to our proof-of-concept implementations of our hypothesized attacks. By publicly releasing our software enhancements and dataset of captured Thread packets, we hope that we will enable more researchers to study the security of Thread networks, including the ones that Matter-enabled Thread devices will form when they will be available as commercial smart home products.

ACKNOWLEDGMENTS

This research was supported in part by the Secure and Private IoT initiative (IoT@CyLab) at the Carnegie Mellon CyLab Security and Privacy Institute.

REFERENCES

[1] Adafruit Industries. [n. d.]. *Adafruit Feather nRF52840 Express.* Retrieved February 12, 2022 from https://www.adafruit.com/product/4062

[2] Dimitrios-Georgios Akestoridis. [n. d.]. *A collection of GNU Radio Companion flow graphs for the inspection of IEEE 802.15.4-based networks.* Retrieved February 12, 2022 from https://github.com/akestoridis/grc-ieee802154

[3] Dimitrios-Georgios Akestoridis. [n. d.]. *Modified ATUSB firmware that supports selective jamming and spoofing attacks.* Retrieved February 12, 2022 from https://github.com/akestoridis/atusb-attacks

[4] Dimitrios-Georgios Akestoridis. [n. d.]. *Zigator: A security analysis tool for Zigbee and Thread networks.* Retrieved March 26, 2022 from https://github.com/akestoridis/zigator

[5] Dimitrios-Georgios Akestoridis, Madhumitha Harishankar, Michael Weber, and Patrick Tague. 2020. Zigator: Analyzing the Security of Zigbee-Enabled Smart Homes. In *Proceedings of the 13th ACM Conference on Security and Privacy in Wireless and Mobile Networks (WiSec).* 77–88. https://doi.org/10.1145/3395351.3399363

[6] Dimitrios-Georgios Akestoridis and Patrick Tague. 2021. CRAWDAD dataset cmu/zigbee-eda (v. 2021-10-22). https://doi.org/10.15783/t8mt-a674

[7] Dimitrios-Georgios Akestoridis and Patrick Tague. 2021. HiveGuard: A Network Security Monitoring Architecture for Zigbee Networks. In *Proceedings of the 2021 IEEE Conference on Communications and Network Security (CNS).* 209–217. https://doi.org/10.1109/CNS53000.2021.9705043

[8] 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

[9] Bastian Bloessl. [n. d.]. *IEEE 802.15.4 ZigBee Transceiver.* Retrieved February 12, 2022 from https://github.com/bastibl/gr-ieee802-15-4

[10] Bastian Bloessl. [n. d.]. *Some GNU Radio blocks that I use.* Retrieved February 12, 2022 from https://github.com/bastibl/gr-foo

[11] 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.

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

[13] Blake D. Bryant and Hossein Saiedian. 2017. A novel kill-chain framework for remote security log analysis with SIEM software. *Computers & Security* 67 (2017), 198–210. https://doi.org/10.1016/j.cose.2017.03.003

[14] Xianghui Cao, Devu Manikantan Shila, Yu Cheng, Zequ Yang, Yang Zhou, and Jiming Chen. 2016. Ghost-in-ZigBee: Energy Depletion Attack on ZigBee-Based Wireless Networks. *IEEE Internet of Things Journal* 3, 5 (2016), 816–829. https://doi.org/10.1109/JIOT.2016.2516102

[15] Connectivity Standards Alliance. [n. d.]. *Amazon, Apple, Google, and the Alliance and Its Board Members Form Industry Working Group to Develop a New, Open Standard for Smart Home Device Connectivity.* Retrieved February 12, 2022 from https://csa-iot.org/newsroom/connectedhomeip/

[16] Connectivity Standards Alliance. [n. d.]. *Connectivity Standards Alliance Matter Update.* Retrieved March 23, 2022 from https://csa-iot.org/newsroom/matter-march-update/

[17] Connectivity Standards Alliance. [n. d.]. *The Connectivity Standards Alliance Unveils Matter, Formerly Known as Project CHIP.* Retrieved February 12, 2022 from https://csa-iot.org/newsroom/chip-is-now-matter/

[18] Connectivity Standards Alliance. [n. d.]. *Matter (formerly Project CHIP) is creating more connections between more objects, simplifying development for manufacturers and increasing compatibility for consumers, guided by the Connectivity Standards Alliance (formerly Zigbee Alliance).* Retrieved February 12, 2022 from https://github.com/project-chip/connectedhomeip

[19] Connectivity Standards Alliance. [n. d.]. *Zigbee.* Retrieved February 12, 2022 from https://csa-iot.org/all-solutions/zigbee/

[20] CRAWDAD. [n. d.]. *CRAWDAD: A Community Resource for Archiving Wireless Data At Dartmouth.* Retrieved March 25, 2022 from https://crawdad.org/

[21] Daniel Dinu and Ilya Kizhvatov. 2018. EM Analysis in the IoT Context: Lessons Learned from an Attack on Thread. *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2018, 1 (2018), 73–97. https://doi.org/10.13154/tches.v2018.i1.73-97

[22] Morris Dworkin. 2007. *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.

[23] Ettus Research. [n. d.]. *USRP N210 Software Defined Radio (SDR).* Retrieved February 12, 2022 from https://www.ettus.com/all-products/un210-kit/

[24] GNU Radio. [n. d.]. *GNU Radio – the Free and Open Software Radio Ecosystem.* Retrieved February 12, 2022 from https://github.com/gnuradio/gnuradio

[25] Feng Hao. 2017. J-PAKE: Password-Authenticated Key Exchange by Juggling. RFC 8236. https://doi.org/10.17487/rfc8236

[26] Feng Hao. 2017. Schnorr Non-interactive Zero-Knowledge Proof. RFC 8235. https://doi.org/10.17487/rfc8235

[27] Jonathan W. Hui and Pascal Thubert. 2011. Compression Format for IPv6 Datagrams over IEEE 802.15.4-Based Networks. RFC 6282. https://doi.org/10.17487/rfc6282

[28] Eric M. Hutchins, Michael J. Cloppert, and Rohan M. Amin. 2010. Intelligence-Driven Computer Network Defense Informed by Analysis of Adversary Campaigns and Intrusion Kill Chains. White Paper. Retrieved February 12, 2022 from https://www.lockheedmartin.com/content/dam/lockheed-martin/rms/documents/cyber/LM-White-Paper-Intel-Driven-Defense.pdf

[29] IEEE Computer Society. 2006. IEEE Standard for Information technology—Telecommunications and information exchange between systems—Local and metropolitan area networks—Specific requirements—Part 15.4: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (WPANs). IEEE Std 802.15.4-2006. https://doi.org/10.1109/IEEESTD.2006.232110

[30] Chris Karlof and David Wagner. 2003. Secure Routing in Wireless Sensor Networks: Attacks and Countermeasures. *Ad Hoc Networks* 1, 2 (2003), 293–315. https://doi.org/10.1016/S1570-8705(03)00008-8

[31] Richard Kelsey. 2015. *Mesh Link Establishment.* Internet-Draft draft-ietf-6lo-mesh-link-establishment-00. Internet Engineering Task Force. Retrieved February 12, 2022 from https://datatracker.ietf.org/doc/html/draft-ietf-6lo-mesh-link-establishment-00

[32] Hyung-Sin Kim, Sam Kumar, and David E. Culler. 2019. Thread/OpenThread: A Compromise in Low-Power Wireless Multihop Network Architecture for the Internet of Things. *IEEE Communications Magazine* 57, 7 (2019), 55–61. https://doi.org/10.1109/MCOM.2019.1800788

[33] Hugo Krawczyk, Mihir Bellare, and Ran Canetti. 1997. HMAC: Keyed-Hashing for Message Authentication. RFC 2104. https://doi.org/10.17487/rfc2104

[34] Yu Liu, Zhibo Pang, György Dán, Dapeng Lan, and Shaofang Gong. 2018. A Taxonomy for the Security Assessment of IP-Based Building Automation Systems: The Case of Thread. *IEEE Transactions on Industrial Informatics* 14, 9 (2018), 4113–4123. https://doi.org/10.1109/TII.2018.2844955

[35] Gabriel Montenegro, Nandakishore Kushalnagar, Jonathan W. Hui, and David E. Culler. 2007. Transmission of IPv6 Packets over IEEE 802.15.4 Networks. RFC 4944. https://doi.org/10.17487/rfc4944

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

[37] National Institute of Standards and Technology. 2002. *Secure Hash Standard (SHS).* FIPS 180-2.

[38] OpenThread. [n. d.]. *Build a Thread network with nRF52840 boards and OpenThread.* Retrieved February 12, 2022 from https://openthread.io/codelabs/openthread-hardware

[39] OpenThread. [n. d.]. *IPv6 Addressing.* Retrieved February 12, 2022 from https://openthread.io/guides/thread-primer/ipv6-addressing

[40] OpenThread. [n. d.]. *Network Discovery and Formation.* Retrieved February 12, 2022 from https://openthread.io/guides/thread-primer/network-discovery

[41] OpenThread. [n. d.]. *Node Roles and Types.* Retrieved February 12, 2022 from https://openthread.io/guides/thread-primer/node-roles-and-types

[42] OpenThread. [n. d.]. *OpenThread on Nordic nRF528xx examples.* Retrieved February 12, 2022 from https://github.com/openthread/ot-nrf528xx

[43] OpenThread. [n. d.]. *OpenThread released by Google is an open-source implementation of the Thread networking protocol.* Retrieved February 12, 2022 from https://github.com/openthread/openthread

[44] Qi Hardware Inc. [n. d.]. *Ben-WPAN Overview.* Retrieved February 12, 2022 from http://downloads.qi-hardware.com/people/werner/wpan/web/

[45] Eric Rescorla and Nagendra Modadugu. 2012. Datagram Transport Layer Security Version 1.2. RFC 6347. https://doi.org/10.17487/rfc6347

[46] 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

[47] SecDev. [n. d.]. *Scapy: the Python-based interactive packet manipulation program & library.* Retrieved February 12, 2022 from https://github.com/secdev/scapy

[48] Thread Group. [n. d.]. *Thread Group: Member Benefits.* Retrieved February 12, 2022 from https://www.threadgroup.org/thread-group#Membershipbenefits

[49] Thread Group. [n. d.]. *What is Thread: Overview.* Retrieved February 12, 2022 from https://www.threadgroup.org/What-is-Thread/Overview

[50] Thread Group. 2015. Battery-Operated Devices. White Paper. Retrieved February 12, 2022 from https://www.threadgroup.org/Portals/0/documents/support/BatteryOperatedDevicesWhitePaper_656_2.pdf

[51] Thread Group. 2015. Thread Commissioning. White Paper. Retrieved February 12, 2022 from https://www.threadgroup.org/Portals/0/documents/support/CommissioningWhitePaper_658_2.pdf

[52] Thread Group. 2015. Thread Usage of 6LoWPAN. White Paper. Retrieved February 12, 2022 from https://www.threadgroup.org/Portals/0/documents/support/6LoWPANUsage_632_2.pdf

[53] Thread Group. 2020. Thread Network Fundamentals. White Paper. Retrieved February 12, 2022 from https://www.threadgroup.org/Portals/0/documents/support/Thread%20Network%20Fundamentals_v3.pdf

[54] Wireshark Foundation. [n. d.]. *Wireshark's official Git repository.* Retrieved February 12, 2022 from https://gitlab.com/wireshark/wireshark
[55] Anthony D. Wood and John A. Stankovic. 2002. Denial of Service in Sensor Networks. *Computer* 35, 10 (2002), 54–62. https://doi.org/10.1109/MC.2002.1039518
[56] 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
[57] Zigbee Alliance. 2016. *Base Device Behavior Specification.* ZigBee Document 13-0402-13.
[58] Tobias Zillner and Sebastian Strobl. 2015. ZigBee Exploited - The Good, the Bad and the Ugly. Presented at Black Hat USA 2015.