Disconnection Attacks Against LoRaWAN 1.0.X ABP Devices

1st Giorgio Bernardinetti
CNIT
University of Rome "Tor Vergata"
Rome, Italy
giorgio.bernardinetti@cnit.it

2nd Francesco Mancini
CNIT
University of Rome "Tor Vergata"
Rome, Italy
francesco.mancini@cnit.it

3rd Giuseppe Bianchi
CNIT
University of Rome "Tor Vergata"
Rome, Italy
giuseppe.bianchi@uniroma2.it

***Abstract*—Previous research work has already documented vulnerabilities of LoRaWAN 1.0.x, in the form of *Replay Attacks* which may cause disconnection situations. To face (also) these concerns, modern network servers implement careful techniques to handle sequence numbers (frame counters) in the presence of unexpected/out-of-sequence messages. In this paper we show that, despite such patches, the problem of disconnection attacks is still widely open. We document a number of new replay-type attacks which target ABP (Activation By Personalization) devices, namely devices which are deployed with an hard-coded set of session keys, and which may cause a range of disconnection situations, including extremely long term ones - the worst case being in the order of 2 to the 32 message transmissions (hundreds/thousands years considering ordinary IoT rates). We demonstrate the feasibility of the proposed attacks by analyzing their impact on three different LoRaWAN network server implementations (two well known open-source network servers, and a proprietary network server co-developed by us), and demonstrating their practicality on two of said network servers (ours and ChirpStack). Finally, we discuss trade-offs and mitigation actions, though we remark that these attacks appear intrinsic in the LoRaWAN 1.0.x specification, and can be ultimately fixed only by migrating to LoRaWAN 1.1.**

I. INTRODUCTION

Internet of Thing (IoT) is becoming widely adopted thanks to the deployment of IoT devices in every corner such as economy, industry, wearables and domotics. The huge (order of many billions) number of connected objects, their limited resources, and the fact that they operate autonomously without a human involvement emphasizes the importance of security in ways that cannot be ignored.

In the context of IoT, this paper specifically focuses on LoRaWAN [1], an emerging Low Power Wide Area Network (LPWAN) technology. LoRaWAN is a MAC-layer protocol based on the radio protocol LoRa [2], a wireless physical layer solution developed by the company Semtech for long-range low-powered, low-data-rate applications.

Security of IoT, in terms of either availability of the network as well as message integrity and protection, is a crucial requirement in sight of the possible damage that unauthorized message tampering or massive denial of service attacks may cause, and LoRaWAN makes no exception. Indeed, the security of LoRaWAN has been thoroughly scrutinized in the recent literature, and the reader interested in a detailed analysis of the various attack types and vulnerabilities may refer to [3]–[6], among the others.

Our work starts from the results presented in [3]. In this paper, Yang et. al. document a LoRaWAN 1.0 vulnerability in the handling of sequence numbers. As reviewed later on in Section III-A, they show that an attacker may easily perform a *Replay attack*, and exploit a carefully selected replayed message to cause a temporary denial of service in the form of temporary disconnection of a targeted device. Even if the actual proof-of-concept implementation of the attach showcased in [3] is not fully practical, as it relies on supplementary manual configurations (details in section III-A), the vulnerability appears very critical, to the extent that most of the deployed network servers have devised tailored solutions to mitigate such an attack.

In this paper, with concrete reference to three modern network server implementations, we show that the problem is not nearly solved by the so-far considered custom patches.

To this purpose, we focus on two proposed approaches also devised to cope with the attack presented in [3], one (section III-B) implemented in a network server we have co-developed and deployed in the Rome metropolitan area, and a second one (section III-C) implemented in two well known open source network servers, ChirpStack and TTN (The Things Network). We show that this second approach, seemingly more clever than our own implementation, actually suffers from a technical flaw which may lead to a practically persistent disconnection situation even without any external attacks.

In addition, we discuss new forms of *Replay Attacks* which permit an attacker to disconnect target devices from the network servers, by de-synchronizing the relevant frame counters. Also depending on the network server configuration and implementation, our proposed attacks have a varying severity, ranging from "temporary" (order of tens or hundreds of messages lost) up to - in practice - "persistent": the worst case disconnection period may even reach a value in the order of $2^{32} - 2^{14} \approx 4.3$ billion message transmissions (e.g. 136 years even assuming a very high transmission rate of $1\ message/s$) - re-connection is made possible only by explicit human intervention on the network server.

In summary, this paper contributes as follows:

- With a number of newly proposed replay-type attacks, we unveil the subtleties underlying the techniques devised to solve a previously documented LoRaWAN 1.0 vulnerability [3];
- We discuss the feasibility of the attacks for three different network server implementations: the ChirpStack

open source network server, the open source network server from TTN - The Things Network - and a proprietary network server co-developed by us and currently deployed in the Rome metropolitan area;
- We unveil a design bug in the technique currently adopted in either the ChirpStack network server as well as in TTN, showing that a persistent disconnection situation may emerge also for natural causes, i.e. even without the intervention of an external attacker - though we show how an attacker may easily accelerate the emergence of such situations.
- We further experimentally demonstrate the practical feasibility of the attacks using as reference network server implementation the one from ChirpStack (but we remark, as discussed in more details in the dedicated section, that the same experiment would work also on the other considered platforms);
- We finally discuss trade-offs and (partial) mitigation techniques - partial because the vulnerability appears to be non-fixable and intrinsic in the LoRaWAN 1.0 protocol specification [8]. A full mitigation of the attack appears in fact to require migration to the next version 1.1 [10] of the LoRaWAN specification.

The rest of this paper is organized as follows. Section II introduces the security features of the LoRaWAN protocol; Section III describes the vulnerability and how it can be exploited; Section IV shows the practical feasibility of the attack; Section V is an overview of related work; Section VI concludes this paper.

II. SECURITY FEATURES OF LORAWAN

In this section an overview of the security features of the LoRaWAN protocol is presented. More specifically, we focus on the protocol version 1.0.3 [8], which is the most recent specification of LoRaWAN 1.0.X. Being based on wireless omnidirectional communication and ultra-low-cost equipments, LoRaWAN deployments may be very easily tampered by attackers capable of sniffing, injecting and modifying messages. Any protocol seeking to provide security services in LoRaWAN deployments must take care about these issues, and provide support for techniques aimed to provide Confidentiality, Integrity and Availability of the network. In addition, to provide end-device authentication and to establish the connection security parameters, a network access method should be provided by the protocol. In the following we will briefly review how LoRaWAN addresses these security objectives.

A. Security parameters

LoRaWAN protocol defines two session keys for confidentiality and integrity: Network Session Key (NwkSKey) and Application Session Key (AppSKey). The first one is aimed to handle secure communication between the end-device and the network server (Network operator infrastructure), while the latter guarantees end-to-end security between the end-device and third party application. The reason behind these two keys is that LoRaWAN was designed with the goal of decoupling the network operator, which provides LoRaWAN infrastructure, and the end-user who uses the network as a black-box to deploy custom applications among IoT devices.

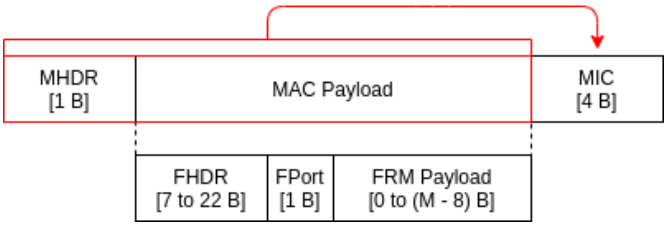

Fig. 1. MIC is computed on MAC header and payload

The protocol specifies two mechanisms in order to distribute the session keys: *Activation By Personalization (ABP)* and *Over-The-Air Activation (OTAA)*

1) Activation By Personalization (ABP): In an ABP device the session keys are hardcoded inside the firmware and stored on the network server side; these keys can only be changed by manually flashing the firmware and by changing the values stored in the network server database. When an ABP device wants to connect to the network infrastructure, there is no need for an initial handshake, because the security parameters are already defined.

2) Over-The-Air Activation (OTAA): In the case of an OTAA device, there is an initial handshake procedure, called *Join Procedure*, before exchanging data. The aim of this procedure is to establish the security parameters for the session (NwkSKey, AppSKey). The security aspects of this procedure are guaranteed by using a pre-shared key, called Application Key (AppKey). The end-device sends a *Join Request* message, which contains a 2-bytes random number, called DevNonce; it is sent in clear and its integrity is guaranteed by a Message Integrity Code (MIC), as explained in the next section II-B. When the network server receives a *Join Request* message, it replies to the request only if i) the integrity check is valid, and ii) the DevNonce is not repeated. In this case, the Network Server starts the Join Procedure by replying with a *Join Accept* message, which contains a 3-bytes random number, called AppNonce, and the LoRaWAN address assigned to it, called DevAddr; the message is sent encrypted and its integrity is guaranteed by MIC. The device will accept the message only if it is valid (i.e. integrity check OK and AppNonce not repeated). The two parties separately generate the session keys by executing AES encrypt operation in ECB mode on exchanged data and AppKey.

B. Integrity

Message integrity is provided by a message integrity code (MIC) which is applied to both the MAC header and the payload of the message as shown in Figure 1.

The MIC is computed by applying the AES128-CMAC protocol; the key used depends on the type of message: MIC on Join Request and Join Accept messages are generated using the AppKey; otherwise, they are generated with the NwkSKey.

C. Frame counters

Each uplink and downlink message is tagged with a sequence number, called $FCntUp$ and $FCntDown$ respectively. The goal of these values is to protect the system

from *Replay Attacks*. Frame counters are reset to 0 after a successful Join Procedure or a reset of a personalized end-device. They are incremented at the sender side by 1 for each new data frame sent in the respective direction. The receiver will accept only messages whose counter is higher than the receiver stored one. In addition, the difference between frame counter in the received message and stored one must be lower than MAX_FCNT_GAP (i.e. 16,384).

III. DEVICE DISCONNECTION ATTACKS

The attacks documented in the remainder of this section are based on a different and novel way to exploit message *replay* opportunities so as to create de-synchronization situations which cause the disconnection of a target device from the network. To this purpose, we start from reviewing in section III-A the attack proposed in [3]. The core of our work is discussed in the next two sections III-B and III-C. Here, after reviewing two different mechanisms we have found implemented in practical network servers to mitigate such a baseline vulnerability, we present our own attacks and show how they can make such countermeasures ineffective.

More specifically, section III-B shows how the decision to reset *both the uplink and downlink* frame counters after an out-of-order message (following in spirit the LoRaWAN 1.0.X provisions for coping with device resets - see next section III-A) still permits a temporary device disconnection. Conversely, section III-C shows that the apparently more clever solution to reset *only the uplink* frame counter - indeed a strategy that we found implemented in the two open source network server implementations we have analyzed, is actually more nefarious, as it opens the door not only to de-synchronization attack yielding a (practically) persistent disconnection periods, but it may also cause disconnection situations for "natural" causes - i.e. even without any attack.

A. *The Yang et. al [3] baseline* Replay Attack

Paper [3] has documented a very simple *Replay Attack* which can be used to cause temporary disconnection of ABP (Activation By Personalization) devices. A main drawback of ABP devices is the lack of persistent storage of the latest value of the frame counters. This implies that an ABP device, once reset (rebooted), restarts all its frame counters from 0. Therefore, they reset uplink and downlink counters when they are switched off. If this happens, an ABP device can no longer communicate with the LoRaWAN Network server: the end-node sends uplink messages with $FCntUp$ value lower than the Network Server one; according to the protocol, such a message will be rejected until its $FCntUp$ value will reach the one expected by the server. Hence, these devices disconnect from the network after a reboot. It readily follows that a reset would cause a device to send messages that reuse previous sequence numbers, i.e., that to the perspective of the network server might appear spoofed or replayed. To cope with this (potentially frequent) event, the LoRaWAN 1.0.X protocol states what follows:

> *After a JoinReq – JoinAccept message exchange or a reset for a personalized end-device, the frame counters on the end-device and the frame counters on the network server for that end-device are reset to 0*

.

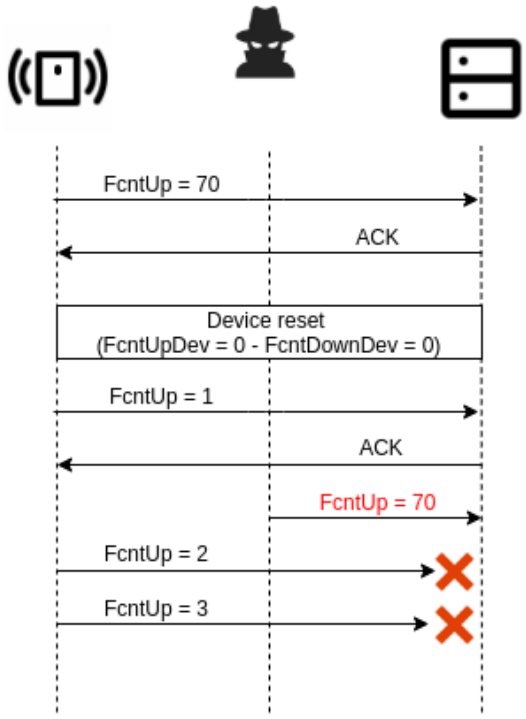

Fig. 2. Denial-of-Service on an ABP device exploiting the *Replay Attack*.

As shown in [3], and graphically illustrated in the example of figure 2, this protocol choice opens a potential *Replay Attack* vulnerability, as a message intercepted before the device reset can now be *replayed* to cause a temporary outage. More specifically, in figure 2 we assume that the attacker has stored message #70. After this message, the device is reset. Following the LoRaWAN 1.0.X provisions, the server is expected to also reset the frame counters[1], so as to accept messages restarting from the value #0.

At this point, the attacker may perform a trivial *Replay Attack*, and retransmit the previously stored message #70. This message is duly authenticated (as it is replayed from the original device), and therefore the network server takes it as a valid one, and updates the uplink counter to the value #70. The result is that all the *legitimate* messages sent by the device will be discarded, until $FCntUp$ will reach again the value #70.

B. *Attacks to the "reset-both-counters" approach*

An apparently obvious way to counter the previously described attack consists in further generalizing the conditions under which the frame counters on both ends are reset, and specifically *reset the frame counters not only when an explicit device reset is notified, but also whenever any out-of-sequence message is received* - i.e. in practice assume that an out of sequence message implies that a device reset has happened and follow the relevant standard provisions.

[1]However, it is not obvious how the server may automatically recognize that a device is reset - to this end, [3]'s proof-of-concept apparently relies on the assumption that the LoRaWAN network infrastructure is notified out-of-band (e.g. by human intervention) when an ABP device is reset.

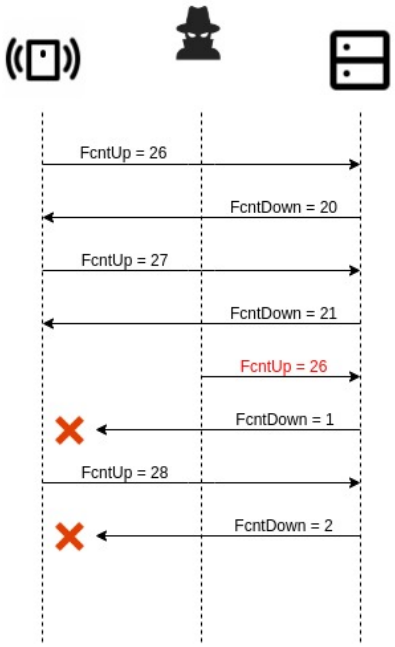

Fig. 3. *Replay Attack* with reset of both frame counters.

At the expense of reduced session integrity[2], this solution might seem to be a valid patch against the disconnection attack [3] described in the previous section. Indeed, with reference to the previous figure 2, after the replayed message #70, the subsequent message #2 would cause a reset of the network server frame counters, and therefore would permit the server to accept this and all the subsequent messages.

However, a closer look reveals that if, generalizing the provisions of the standard also to the case of out-of-sequence messages, we reset both network server side frame counters, then a *different* disconnection attack may still be carried out, this time in the opposite direction.

The attack against this network server implementation choice is illustrated in figure 3. The attack consists in replaying a previously captured message, for instance message #26 in the figure, to the network server. This out of order message will cause the network server to reset *both* uplink and downlink counters to 0. The next legitimate uplink message, namely #28, will restore the proper sequence on the uplink direction, which will therefore *not* suffer of any outage. However, the *downlink* counter will now be lower than what the device expects (in the figure, downlink message #22), and will therefore be discarded by the device[3], along with all the subsequent messages until the downlink counter reaches again the original value #22.

The consequence is an half-disconnection: the server will be temporarily unable to communicate with the device since the $FCntDown$ in the downlink message remains lower than the one on the device side - this can be especially critical in front of unacknowledged downlink communication, as the server might not be able to understand that a disconnection

[2]an attacker may in fact inject replayed messages which the network server will not recognize anymore as spoofed ones.

[3]while a network server patch is doable, we of course do not consider viable the possibility to patch *all* the worldwide devices to *also* accept out of order messages!

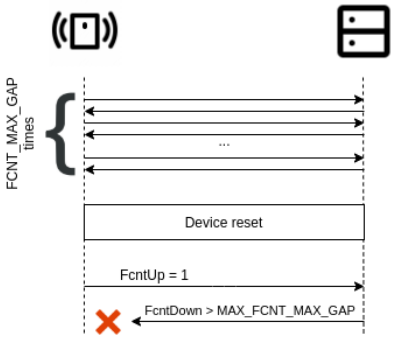

Fig. 4. Device behaviour when the downlink counter is not reset

has occurred. Note that the time needed in order to re-synchronize the frame counters is roughly equal to the amount of time the device was powered on. This is because the Network Server can send a downlink message only when it receives an uplink one (class-A devices), so the downlink message rate is bounded to the uplink one. Considering a device sending 70 messages in a period of 70 days, its message rate is $MsgRate = 1\ message/day$ and its downlink counter value is equal to 70; after a successful attack, the network server will take $\frac{FCntDown}{PktRate} = 70$ days to re-synchronize the downlink counter.

Recover from the attack: Apparently, the only ways to instantly recover from this disconnection attack imply manual intervention in one of these two forms:

1) Manually reset the device; this requires physical access to the device.
2) Restore the last $FCntDown$ value sent by the server to the device before the attack in the database; this requires the knowledge of such value, so the server should keep track of the downlink messages history.

C. Attacks to the "reset-only-uplink-counter" approach

In the previous section, we have clearly understood that the reset of both frame counters is not a valid approach. The reader might then argue that a better approach consists in resetting only the frame counter affected by the actual out-of-order, namely the uplink counter, and more specifically set it equal to the value of the out-of-order offending message. In fact, this is what is done by both the open source implementations whose source code we have analyzed, namely *ChirpStack*'s LoRaWAN Network Server stack (v3.7.0) and *The Things Network* (v2.10.3).

However, as detailed in what follows, a closer look reveals two crucial issues. First, this approach does not solve disconnection attacks. Second, and perhaps even more critical, this approach appears to bring about serious technical concerns (it actually looks to us as a technical flaw), as a persistent outage situation may eventually occur just for *accidental* reasons, namely a legitimate device reboot - in other words, we do not even need any explicit attack to create a de-synchronization situation between end-device and network server.

Let us start by illustrating this technical flaw, and then extend the analysis to the attack case. First, we recall that the LoRaWAN 1.0.X standard defines a "receiver window" expressed as the maximum acceptable difference between

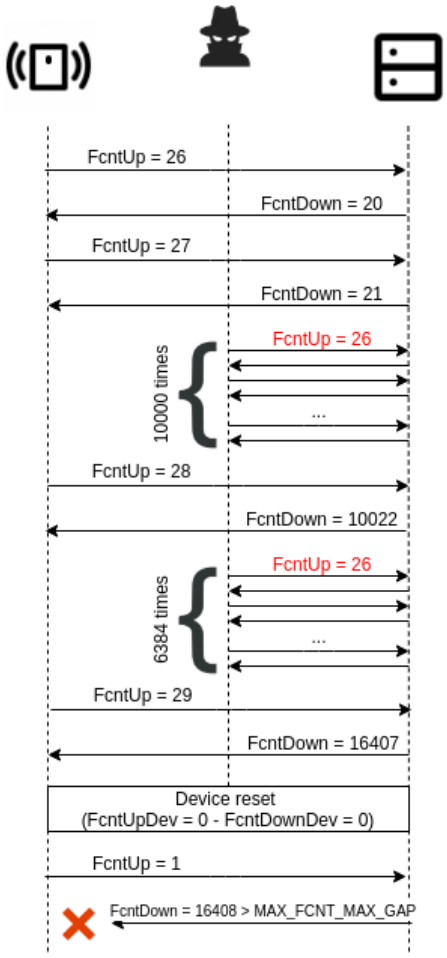

Fig. 5. Speed up of the attack.

the value of a received counter and a stored one. Specifically, if the difference between the received and stored counters overflows the constant MAX_FCNT_GAP (by default configured to the value 16,384), then the message is discarded. Now the decision of *never* resetting the downlink counter may eventually lead to a large value of said counter, and specifically to a value larger than MAX_FCNT_GAP. Assume now that the ABP device incidentally reboots, for autonomous reasons. It will restart with both uplink and downlink counters reset to zero. But then, when the device transmits the first message after the reboot, the server will only reset the uplink counter, and will therefore reply with the current value of the downlink counter - if this is larger than MAX_FCNT_GAP, the message will be discarded by the device, *along with all subsequent messages!* - this scenario is depicted in Figure 4.

To the best of our knowledge, this technical bug (which we have also experimentally double-checked, see next section) has never been disclosed so far. Perhaps one of the reasons is that the above situation might appear quite infrequent, as it would require a device to send as many as 16,384 messages and then[4] reset - even assuming 1 message per hour which would be an extremely unusual large rate for ABP devices,

[4]Actually, it is not necessarily that the device remains active for the whole transmission period of the 16,384 messages: the device might in fact also reboot multiple times in the meantime as the server in any case continues to increase the downlink counter.

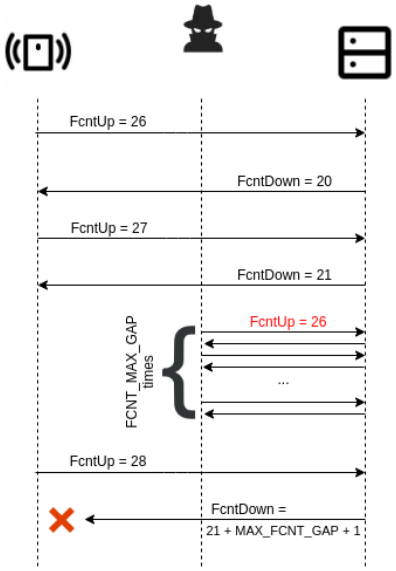

Fig. 6. Final version of the attack.

this would take almost 2 years! However, as shown in figure 5, an attacker can trivially *accelerate* such a situation by replaying any arbitrary valid device message multiple times, thus significantly accelerating the increase of the downlink counter.

Finally, under more strict conditions, the attacker may even perform a de-synchronization scenario which *does not require explicit reset of the device*. This attack is illustrated in figure 6. In this case, the attacker must be able to inject MAX_FCNT_GAP replayed messages during one *single* interval between two successive "legitimate" transmissions, This would cause the difference between the downlink counter sent by the server and the one stored at the device to be greater than MAX_FCNT_GAP, and thus cause a persistent disconnection. The device will in fact discard any subsequent downlink messages because $FCntDown_{Nwk} - FCntDown_{Dev} \geq MAX\_FCNT\_GAP$, and this gap will increase at every further message transmission, until the frame counter will recirculate - considering that the frame counters are based on 32 bit registers, without any manual intervention, the number of messages necessary to re-synchronize and therefore restore downlink connectivity is equal to $2^{32} - MAX\_FCNT\_GAP \approx 4.3$ billion messages, e.g. 136 years even assuming a very high transmission rate of $1\ message/s$.

D. Discussion and Caveats

A first question is whether the last attack described in the previous section is practical; in other words, is it viable to spoof as many as MAX_FCNT_GAP replayed messages in the period of time between two subsequent messages transmitted by the legitimate device? As a matter of fact, this is a main requirement in order for the attack to be successful: a number of spoofed messages smaller than MAX_FCNT_GAP would permit the end-device to "synchronize" with the new downlink counter - the device would in fact receive a message with a counter value in

the acceptable window and would therefore update the local counter state accordingly.

To answer the above question, we make an estimate of the amount of time it might take for an attacker to replay such a massive number of messages, i.e. MAX_FCNT_GAP message. Given a message of 30 bytes, a spreading factor value (see [9] for more information) equal to 7 (i.e. $5,470\ bit/s$) and 2 seconds delay for processing the message (usually the Network server must send the downlink message up to 2 second after the end of the corresponding uplink one transmission), it would take theoretically $((30 * 8)/5,470 + 2) * 16,384s \approx 9h20m$. Because of this estimate, the target end-device should have a message-rate of at least 1 $message$ every 10 $hours$. In a real scenario, this assumption makes such an attack practical, because end-devices usually send 1 or 2 messages per day.

Moreover, if the above transmission rate of spoofed messages is not affordable by the attacker, we point out that it is still possible to replay messages across *multiple* transmission periods, provided that the intermediate messages are properly jammed, for instance using the *jamming* attack described in [7], so as to avoid that the device re-aligns its downlink counter state.

Finally, Jamming may also be exploited to extend the attack to class C devices. Indeed, if the device belongs to class C, it is always listening to the network server response messages and thus it would also overhear the server replies to the malicious uplink messages, thus increasing its downlink counter value and disabling the attack conditions.

IV. EXPERIMENTAL RESULTS

We have experimentally verified the previously described attack scenarios.

A. network server with "reset-both-counters"

An Adeunis Field Test Device [12] was used as victim device. Any LoRa transceiver capable of receiving and transmitting LoRa packets can be used to perform the attack, so we used a LoRaWAN gateway produced by Kerlink [13] which can both receive and transmit packets.

The gateway was used to intercept uplink messages sent by the target device (identified by its DevAddr parameter) which was configured to send one packet per hour. A custom script was written in order to replay such captured messages after a configurable amount of time (we chose 3 hours).

After the attack, the victim device could not receive downlink messages anymore. However, after sending 3 uplink messages (i.e. after the same amount of time the device was powered on before the attack, as explained in III-B), the device was able to process the next downlink messages sent by the server (and all the subsequent ones) thus restoring its normal functionalities.

B. Network server with "reset-only-uplink-counter": technical flaw

At first, we verified that the technical issue discussed in section III-C (i.e., if the downlink counter value stored in the network server database becomes greater than MAX_FCNT_GAP and the device reboots, it will discard subsequent downlink messages), was actually confirmed by a real world experimentation.

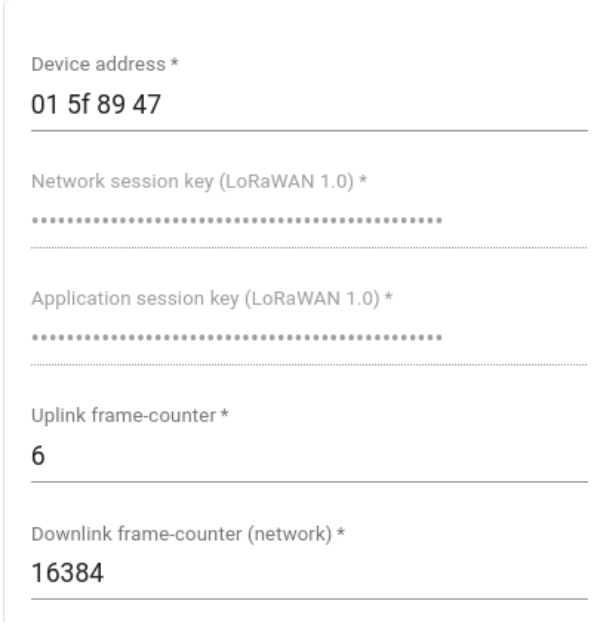

Fig. 7. *Replay Attack* with reset of both frame counters.

We experimentally tested this fact on the *ChirpStack* LoRaWAN Network Server (the TTN server has the very same behavior). An Adeunis Field Test Device [12] was registered in the server as an ABP device and the flag to enable counter reset was checked as it is shown in Figure 8. To save time, we manually configured the downlink counter value to $MAX\_FCNT\_GAP = 16,384$ (Figure 7). After a few extra uplink messages, we then manually rebooted the end-device. As predicted, we found that, after reboot, the device lost downlink connectivity "forever", i.,e. it was not able to receive downlink messages anymore.

C. Network server with "reset-only-uplink-counter": attacks

We then tested the last (more comprehensive) attack described in section III-C. It is achievable with any LoRa transceiver as long as it is able to receive and transmit LoRa packets. An Arduino with LoRaWAN EU module by Microchip was registered to a local instance of ChirpStack Network Server as an ABP device and the flag to enable counter reset was checked; it was set up for transmitting a Confirmed Data Up message every 12 hours.

We performed the attack using a LoRaWAN gateway: a custom software was written in order to automate operations. This program analyzes all incoming traffic until it finds a Confirmed Data Up message of the target device (identified by the DevAddr parameter): this type of message guarantees that the network server replies with a downlink message thus increasing the related counter. When the program detects a suitable message, it starts replaying it every 2 seconds, simulating the behaviour of a normal device. The program terminates after sending 16,500 times the same message; a value greater than MAX_FCNT_GAP has been used in order to prevent any lost message.

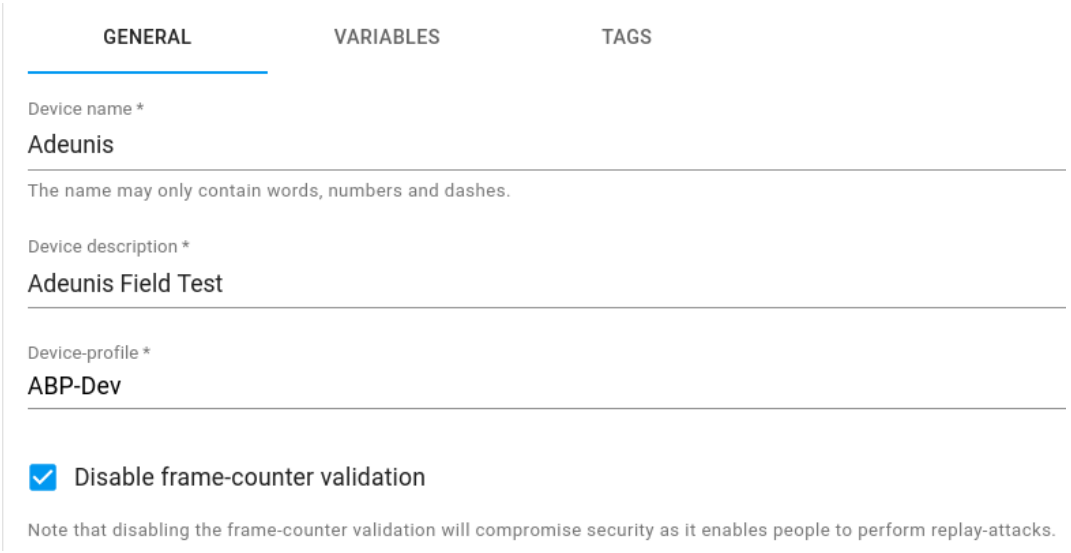

Fig. 8. *Replay Attack* with reset of both frame counters.

Fig. 9. Frame counters values at the server side after the attack.

```
sys get ver
sys factoryRESET
mac set deveui DE39203582271E31
mac set devaddr 013CCF61
mac set nwkskey 542940919D99185664383E0F58F82B04
mac set appskey BDC4A3D9E9D77D19CDCF0E332802EA4A
mac save
mac get deveui
mac get devaddr
sys get ver
mac join abp
mac tx cnf 3 7B2276616C6F7265223A226369616F227D
mac save
Packet correctly acked
Downlink payload received:
        Port: 15
        Data: 6369616F
sys get ver
mac join abp
mac tx cnf 3 7B2276616C6F7265223A226369616F227D
mac save
Error: 5 (Server did not response)
```

Fig. 10. Log of the victim device

```
12/02/2020 18:03:47 [ABP-ATTACK] INFO: ABP Attack Module (ABP-ATTACK) constructor configuration...
12/02/2020 18:03:47 [ABP-ATTACK] INFO: Bootstrapping UDP server...
12/02/2020 18:03:47 [ABP-ATTACK] INFO: UDP Server listening: 0.0.0.0:1680

12/02/2020 18:03:56 [ABP-ATTACK] INFO: Sending 1 packet to Server
12/02/2020 18:03:57 [ABP-ATTACK] INFO: 1 packet sent
12/02/2020 18:03:59 [ABP-ATTACK] INFO: Sending 2 packet to Server
12/02/2020 18:04:00 [ABP-ATTACK] INFO: 2 packet sent
12/02/2020 18:04:01 [ABP-ATTACK] INFO: Sending 3 packet to Server
12/02/2020 18:04:02 [ABP-ATTACK] INFO: 3 packet sent
12/02/2020 18:04:04 [ABP-ATTACK] INFO: Sending 4 packet to Server
12/02/2020 18:04:05 [ABP-ATTACK] INFO: 4 packet sent
12/02/2020 18:04:06 [ABP-ATTACK] INFO: Sending 5 packet to Server
12/02/2020 18:04:07 [ABP-ATTACK] INFO: 5 packet sent
12/02/2020 18:04:09 [ABP-ATTACK] INFO: Sending 6 packet to Server
12/02/2020 18:04:10 [ABP-ATTACK] INFO: 6 packet sent


13/02/2020 05:25:41 [ABP-ATTACK] INFO: Sending 16495 packet to Server
13/02/2020 05:25:42 [ABP-ATTACK] INFO: 16495 packet sent
13/02/2020 05:25:43 [ABP-ATTACK] INFO: Sending 16496 packet to Server
13/02/2020 05:25:44 [ABP-ATTACK] INFO: 16496 packet sent
13/02/2020 05:25:46 [ABP-ATTACK] INFO: Sending 16497 packet to Server
13/02/2020 05:25:47 [ABP-ATTACK] INFO: 16497 packet sent
13/02/2020 05:25:48 [ABP-ATTACK] INFO: Sending 16498 packet to Server
13/02/2020 05:25:49 [ABP-ATTACK] INFO: 16498 packet sent
13/02/2020 05:25:51 [ABP-ATTACK] INFO: Sending 16499 packet to Server
13/02/2020 05:25:52 [ABP-ATTACK] INFO: 16499 packet sent
13/02/2020 05:25:53 [ABP-ATTACK] INFO: Sending 16500 packet to Server
13/02/2020 05:25:54 [ABP-ATTACK] INFO: 16500 packet sent
```

Fig. 11. Program's log of the attack

The result of the attack is depicted in Figure [9]: the network server sent 16,486 downlink messages (2 after receiving proper messages from the target device and 16,484 after receiving replayed messages), exceeding the limit value of $FCNT\_MAX\_GAP$. After the device sends a new message, it rejects the corresponding downlink one because the received frame counter exceeds the protocol limit. This is shown in Figure [10].

The attack took about 11 and a half hours, as shown in Figure 11. The duration of the attack is greater than the theoretical estimate, and the main reason for this is the communication and processing latency between the program and the gateway. Although the attack lasted longer than what was expected, it still remains within the expected limits, demonstrating its feasibility.

V. RELATED WORK

Although there are a number of papers analyzing the security features and vulnerabilities of LoRaWAN, many of which detailing the well-known *Replay Attack*, to date no work has focused on the consequences of the combination of a *Replay Attack* and the frame counters reset of ABP devices which lead to persistent device disconnection from the network.

Yang et al. [3] introduce the *Replay Attack* against ABP devices; They also talk about creating an uplink disconnection (i.e. messages sent by the device are discarded by the network) by exploiting the aforementioned *Replay Attack*; The attack is done by replaying an old captured message after a device is reset and both the device and the network reset the frame counters. This, by protocol, is what should happen upon an ABP device reset; however, it is practically unfeasible because the network doesn't know when a device is reset.

We fill this gap and show how to create a disconnection, which may be persistent, in all the possible scenarios.

Donmez et al. [4] describe LoRaWAN 1.1 and downgrade scenarios of end-devices implementing lower versions of the protocol; the focus is on how to avoid *Replay Attacks* and messages eavesdropping.

Aras et al. ([5] and [7]) present jamming techniques in LoRaWAN to avoid the receipt of messages by devices. They also introduce the *wormhole attack*; We suggest the techniques described here in order to perform our attack against class C devices, which requires jamming.

Finally, Coman [6] analyzes other security aspects of LoRaWAN i.e. how to forge a valid message by brute-forcing the MIC integrity code.

VI. CONCLUSIONS

The feasibility of the attacks shown in this paper demonstrates the need to update LoRaWAN devices to a more recent protocol version. Indeed, we are not aware of patches to LoRaWAN 1.0.3 which can solve in a compelling way the issues and attacks documented in this paper. Obviously every independent network server vendor may improve their monitoring and analytic tasks so as to recognize and thwart an anomalous behavior, such as a sudden increase of the transmission rate of uplink packets. But our take is that a technical solution at the protocol level would be a better approach than delegating the detection of such attacks to the infrastructure monitoring applications.

The update to LoRaWAN 1.1 [10] is certainly a valuable step, as it succeeds in addressing these issues also with device-side modifications. LoRaWAN 1.1 is in fact the new protocol specification for the LoRaWAN network infrastructure. The vulnerability explained in this paper appear to be fixed in two ways, i.e. frame counters are almost never reset and the same value of $FCntUp$ and $FCntDown$ should never be used for different messages. Specifically, the following two quotes from the protocol specification describe the fixes:

- ABP devices have their frame counters initialized to 0 at fabrication. In ABP devices the frame counters MUST NEVER be reset during the device's life time. If the end-device is susceptible of losing power during its life time (battery replacement for example), the frame counters SHALL persist during such event. [10]
- The end-device SHALL NEVER reuse the same $FCntUp$ value with the same application or network session keys, except for retransmission of the same confirmed or unconfirmed frame. The end-device SHALL never process any retransmission of the same downlink frame. Subsequent retransmissions SHALL be ignored without being processed. [10]

When LoRaWAN 1.1 deployments will be ready, our plan for furure work is to more deeply investigate whether these measures are sufficient to prevent disconnection attacks, or whether more subtle vulnerabilities do remain.

ACKNOWLEDGMENT

This work was partially supported by the H2020 SPARTA Project, funded by the European Commission, grant #830892.

REFERENCES

[1] The LoRa Alliance, https://lora-alliance.org/sites/default/files/2018-04/what-is-lorawan.pdf, 2015
[2] LoRa Technology Overview, *https://www.semtech.com/lora*
[3] X. Yang, E. Karampatzakis, C. Doerr, and F. Kuipers, *"Security vulnerabilities in LoRaWAN"* in Proc. IEEE/ACM 3rd Int. Conf. Internet-of-Things Des. Implementation, Orlando, FL, USA, 2018, pp. 129–140.
[4] Tahsin C. M. Donmez, Ethiopia Nigussiea *"Security of LoRaWAN v1.1 in Backward Compatibility Scenarios"*, 2018
[5] E. Aras, G. S. Ramachandran, *"Exploring the Security Vulnerabilities of LoRa"*, 2017
[6] F. L. Coman, K. M. Malarski, M. N. Petersen, S. Ruepp, *"Security Issues in Internet of Things: Vulnerability Analysis of LoRaWAN, Sigfox and NB-IoT"*, 2019
[7] E. Aras, N. Small, G. S. Ramachandran, S. Delbruel, W. Joosen, D. Hughes, *Selective jamming of LoRaWAN using commodity hardware*, 2017.
[8] LoRaWAN® Protocol Specification v1.0.3, *https://lora-alliance.org/sites/default/files/2018-07/lorawan1.0.3.pdf*
[9] LoRaWAN® Regional Parameters v1.0.3, *https://lora-alliance.org/sites/default/files/2018-07/lorawan_regional_parameters_v1.0.3reva_0.pdf*
[10] LoRaWAN® Protocol Specification v1.1, *https://lora-alliance.org/sites/default/files/2018-04/lorawantm_specification_-v1.1.pdf*
[11] RFC8376, Low-Power Wide Area Network (LPWAN) Overview, *https://tools.ietf.org/html/rfc8376*
[12] FTD: network tester - Adeunis *https://www.adeunis.com/en/produit/ftd-network-tester/*
[13] Wirnet iFemtoCel *https://www.kerlink.com/product/wirnet-ifemtocell/*