IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received September 29, 2019, accepted October 27, 2019, date of publication November 4, 2019,
date of current version November 15, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2951425

Covert Channels in the MQTT-Based Internet of Things

ALEKSANDAR VELINOV[1], ALEKSANDRA MILEVA[2], STEFFEN WENDZEL[3,4], AND WOJCIECH MAZURCZYK[5]

[1]Department of Information Technologies, Faculty of Computer Science, University Goce Delcev, 2000 Štip, Macedonia
[2]Department of Computer Engineering and Intelligent Systems, Faculty of Computer Science, University Goce Delcev, 2000 Štip, Macedonia
[3]Department of Computer Science, Worms University of Applied Sciences, 67549 Worms, Germany
[4]Department of Cyber Security, Fraunhofer FKIE, 53177 Bonn, Germany
[5]Division of Software Engineering and Computer Architecture, Warsaw University of Technology, 00-665 Warsaw, Poland

Corresponding author: Aleksandar Velinov (aleksandar.velinov@ugd.edu.mk)

ABSTRACT Network covert channels are a part of the information hiding research area that deals with the secret transfer of information over communication networks. Covert channels can be utilized, for instance, for data leakage and stealthy malware communications. While data hiding in communication networks has been studied within the last years for several major communication protocols, currently no work is available that investigates covert channels for the publish-subscriber model. To fill this gap, we present the first comprehensive study of covert channels in a protocol utilizing the publish-subscriber model, i.e., the Message Queuing Telemetry Transport (MQTT) protocol which is widely deployed in Internet of Things (IoT) environments. In particular, we describe seven direct and six indirect covert channels and we evaluate and categorize them using the network information hiding patterns approach. Finally, in order to prove that MQTT-based covert channels are practically feasible and effective, we implement the chosen data hiding scheme and perform its experimental evaluation.

INDEX TERMS MQTT, network steganography, network covert channels, data hiding, information hiding, IoT.

I. INTRODUCTION

Network information hiding is the discipline that deals with the hidden data transfer over communication networks and its detection. To this end, network information hiding methods using legitimate data flows create so-called covert channels (CCs) that enable concealed data transmission. Many covert channels have been studied for communication protocols within the past three decades [1], [2]. However, to our best knowledge, there is currently no research available on possible covert channels in protocols that use a publish/subscribe model.

Nowadays, many people and organizations are using IoT-devices that operate on the basis of *Message Queuing Telemetry Transport* (MQTT) servers at home or at work. In such setups, MQTT-capable devices can connect to an MQTT server and with its help exchange messages. Additionally, for the sake of automation, it provides a smart hub, which orchestrates all these devices, provides logic and usually a dashboard, through which the user can control all devices, locally or remotely (e.g., via a mobile phone).

The associate editor coordinating the review of this manuscript and approving it for publication was Xiaochun Cheng.

A research performed by Avast in 2018 with the help of the Shodan IoT search engine [3] showed that 49,197 MQTT servers were publicly visible on the Internet due to a misconfigured MQTT protocol, and 32,888 of them had no password protection, meaning that attackers can access them effortlessly and influence messages flowing through them. This can lead to privacy and information leakage in several contexts, ranging from identity theft and detailed observation of inhabitants or office spaces to industrial espionage. The situation is even worse, as most users do not set up access control in their broker configuration. In result, the attacker can even publish on topics available on these servers, thus seizing control of all connected devices (for example, in the smart home scenario). Now, as presented in this paper, these unprotected MQTT servers can be used as innocent accomplices to enable newly created covert channels. Moreover, such MQTT-based covert channels could be used to realize stealthy malware communications as it has been recently reported also for other types of network traffic and environments [4], [5].

That is why, in this paper, we perform a systematic study of potential covert channels for MQTT-based Internet of Things (IoT) environment. In particular, our novel contribution is as follows:

- we propose thirteen network covert channels for MQTT protocol. In more details, we introduce seven direct (i.e., where the covert sender and covert receiver in order to transmit secrets must be active simultaneously) and six indirect covert channels (i.e., where hidden communication parties may not be active at the same time and they typically use an innocent intermediary to transfer secret messages).
- we evaluate and categorize the proposed CCs using network information hiding patterns approach [6] and describe them using a common description method [1]. Moreover, we devise one covert channel that represents previously unknown subpattern (see Section III for moe information on information hiding patterns concept), thus, we contribute to the enhancement of this approach.
- in order to prove that MQTT-based covert channels are feasible and effective in IoT environment we implement the chosen data hiding scheme and perform its experimental evaluation using three performance metrics: bandwidth, robustness, and undetectability.

The rest of the paper is structured as follows. First, in Section II existing works on network data hiding in related domains are presented. Then, in Section III a brief introduction to the network information hiding patterns approach is presented. Next, in Section IV, the most important aspects of MQTT functioning is enclosed, including the description of the publish/subscribe model, types of MQTT connections, different control packets and other MQTT-specific mechanisms. The comprehensive analysis of potential MQTT-based covert channels is conducted in Section V and it contains the main findings about new covert channels in IoT environments which are characterized with the joint pattern-based categorization. In Section VI the properties of the newly defined data hiding methods are discussed, while in Section VII experimental evaluation for the chosen covert channel is included. Finally, Section VIII concludes our work and outlines potential future research directions.

II. RELATED WORK

The security of smart homes and related types of CPS is a rather well-studied topic, see, e.g., [7], [8] for surveys. Several of the smart home/CPS employed communication protocols have been already subjected to fundamental network security analyses, see, e.g., [9]–[11].

Hron [3] from Avast presented how smart homes that deploy MQTT can be hacked. One can connect to an open and unprotected MQTT broker, subscribe to #, and receive all the messages of all the topics for that broker. This means that the attacker can see the status of window sensors, locks, heating/cooling systems, usage of light switches, etc. Location tracking of the mobile devices used for remote control is also possible. If the access control is broken, one can publish in different topics, insert fake data and perform ''replay attacks''. The more devastating attack is when somebody gains access to the unprotected smart hub dashboards (usually when run on the same machine as MQTT server), and can control all attached devices of the MQTT server. Even if the servers and dashboards are protected, insecure SMB shares with passwords in clear form can be discovered and exploited.

Recently, several works have been published that deploy data hiding techniques in some IoT protocols, like several storage covert channels in the Extensible Messaging and Presence Protocol (XMPP) [12], two storage and one timing covert channels in the Building Automation and Control Networking Protocol (BACnet) [13], six storage and two timing covert channels in the Constrained Application Protocol (CoAP) [14], etc. Wendzel *et al.* [15] have shown that one can hide data in a cyber-physical system (e.g., smart building), by slightly modifying some of its components, like sensors, controllers, actuators, etc., as well as by storing secret data in unused registers.

Several papers already exist that propose indirect network covert channels (i.e., those that do not require a direct interaction between the covert sender and the covert receiver), like [16]–[19]. One particularly interesting and recently proposed indirect network storage covert channel that can be utilized within the same LAN was introduced by Schmidbauer *et al.* [20] where two different network protocols: Address Resolution Protocol (ARP) and Simple Network Management Protocol (SNMP) are deployed. The main idea is that the covert sender exploits the ARP cache of an innocent third-party site for storing secret messages, while the covert receiver exploits SNMP protocol for retrieving them. Other indirect ephemeral storage covert channel uses cached entries in the Pending Interest Table (PIT), maintained by a NDN (Named Data Networking) router, together with PIT misses and PIT hits [21].

III. NETWORK INFORMATION HIDING PATTERNS

Wendzel *et al.* [6] have introduced so-called *hiding patterns*, i.e., abstract descriptions of how data can be hidden in network transmissions. Each pattern presents one core idea of how secret data can be represented through network traffic. These patterns form a taxonomy (Fig. 1). The originally proposed taxonomy has been extended a couple of times, with the latest extension being the one of Mazurczyk *et al.* [22]. As can be seen in Fig. 1, hiding patterns divide into two main categories, those that modulate the *timing* behavior of network traffic and those that modulate *storage* values of the network traffic. For instance, timing channels can modify the timing between network packets to encode secret data, while storage channels can modify unused header bits of network packets (among several other methods). Timing channels can be *protocol-agnostic* (their behavior does not take protocol data interpretation into account) or *protocol-aware* (their behavior must consider protocol data interpretation). Storage channels can either modify protocol fields (such as header bits or padding fields) or payload. If protocol fields are modified then this can be done in two ways, either in a *structure modifying* manner, i.e., by extending the protocol header with additional fields, or, in a *structure preserving* way by simply overwriting already existing values in a protocol

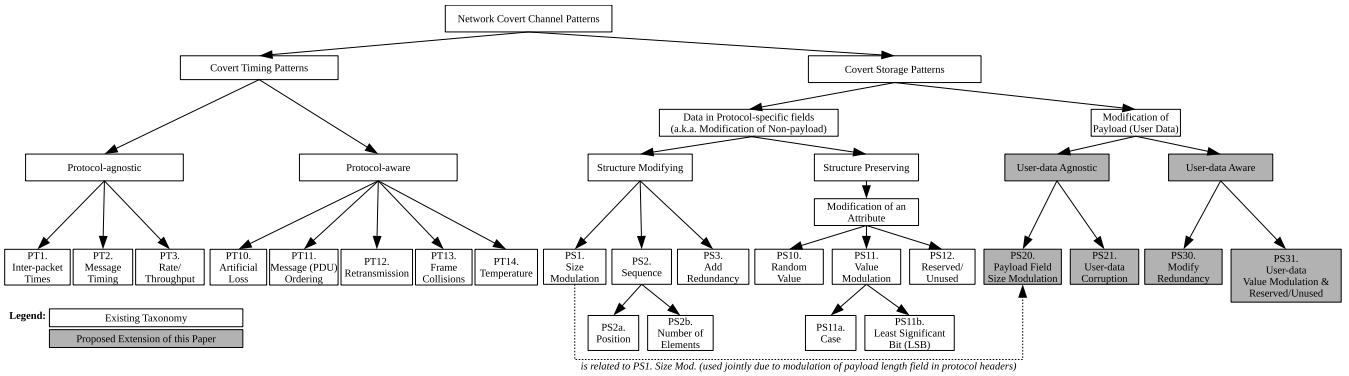

FIGURE 1. Classification of network covert channel patterns (from [22]).

TABLE 1. Network information hiding patterns (extracted from [22]).

Pattern Name	Pattern Description
PT1. Inter-packet Times	The covert channel alters timing intervals between network messages of a flow (interarrival times) to encode hidden data.
PT2. Message Timing	Hidden data is encoded in the timing of message sequences within a flow, e.g. acknowledging every n'th received message or sending commands m times.
PT3. Rate/Throughput	The covert channel sender alters the data rate of a flow from itself or a third party to the covert receiver.
PT10. Artificial Loss	The covert channel signals hidden information via artificial loss of a flow's transmitted messages, e.g. by frame-corruption or message drop.
PT11. Message Ordering	The covert channel encodes data using a synthetic message order in a flow.
PT12. Retransmission	A covert channel retransmits previously sent or received messages of a flow.
PT13. Frame Collisions	The sender causes artificial frame collisions to signal hidden information.
PT14. Temperature	The sender influences a third party node's hardware temperature using traffic of a flow. There must be a technique for the covert receive to measure the temperature (indirectly).
PS1. Size Modulation	The covert channel uses the size of flow metadata (e.g. PDU size or size of a header element) to encode hidden messages.
PS2. Sequence Modulation	The covert channel alters the sequence of flow metadata to encode hidden information. This pattern divides further into: P2.a. Position and P2.b. Number of Elements patterns.
PS3. Add Redundancy	The covert channel embeds redundant metadata (e.g. by adding an unused IP option) in which data is hidden into a flow. Note that in comparison to PS1, the data is hidden in the redundant data's presence, not in the size of an PDU or header element).
PS10. Random Value	The covert channel embeds hidden data into flow metadata that contains a (pseudo-)random value.
PS11. Value Modulation	The covert channel selects one of the n values that a flow's metadata element can contain to encode a hidden message. This pattern divides further into: PS11.a. Case Pattern and PS11.b. Least Significant Bit (LSB) patterns.
PS12. Reserved/Unused	The covert channel encodes hidden data into a flow's reserved or unused metadata elements.
PS20. Payload Field Size Modulation	The size of the payload in a flow is used to encode hidden information (this is a derivate of PS1 but for the payload since it involves the modification of a PDU's payload length field, i.e. PS1).
PS21. User-data Corruption	The covert channel performs a (blind) insertion of covert data into a flow's payload (similar PT10).
PS30. Modify Redundancy	The covert channel compresses a flow's payload and the resulting free space is used to hide data.
PS31. User-data Value Modulation and Reserved/Unused	The covert channel performs a modification of a flow's payload in a way that is not reflected by PS30 and that does not result in a significantly modified interpretation of the data, e.g. by modifying least significant bits of digital images or hiding data in unused/reserved payload bits.

header [1], [6], [22]. Tab. 1 provides an overview of the currently known hiding patterns (the latest version is always available under http://ih-patterns.blogspot.com/).

IV. MQTT FUNDAMENTALS

MQTT is a lightweight, client-server, publish-subscribe message transport protocol, suitable for machine-to-machine (M2M)/IoT connectivity. It is designed for resource-constrained devices and low-bandwidth, high-latency and/or unreliable networks, but it is also suitable for mobile applications. Some previous names of this protocol include: "SCADA protocol", "MQ Integrator SCADA Device Protocol (MQIsdp)" and "WebSphere MQTT (WMQTT)".

MQTT has been heavily applied since its appearance in 1999, e.g., in Facebook Messenger,[1] Amazon IoT (a part of the Amazon Web Services),[2] OpenStack,[3] home automation platform Home Assistant,[4] Microsoft Azure IoT Hub,[5] in the FloodNet project[6] for monitoring river levels and environmental information to provide early warnings of flooding, etc. MQTT libraries are available for many programming

[1]Lucy Zhang: Building Facebook Messenger, 12 August 2011.
[2]https://docs.aws.amazon.com/iot/latest/developerguide/mqtt.html
[3]https://docs.openstack.org/infra/system-config/firehose.html
[4]https://www.home-assistant.io/components/mqtt/
[5]https://docs.microsoft.com/en-us/azure/iot-hub/iot-hub-mqtt-support
[6]http://envisense.org/floodnet/overview.htm

languages and platforms, such as Java, C, C++, C#, JavaScript,.NET, etc.

The MQTT version 3.1.1 [23] became an OASIS standard in 2014 and an ISO/IEC 20922:2016 in 2016. Quite recently, in March 2019, the newest MQTT version 5.0 became an OASIS standard [24]. However, as this version is still very new and rather scarcely deployed (when it comes to software and hardware products), thus, we decided to analyze the version 3.1.1 which is currently most popular. Nevertheless, it must be noted that the study of the covert channels presented in this paper would be in vast majority also applicable to the newest MQTT standard.

A. MQTT CONTROL PACKETS

MQTT v3.1.1 uses 14 different control packets (see Table 2 for their description) and they are numbered from 1 to 14 with maximal size of 256 MB. Every control packet has a fixed header (with the type of the control packet, flags specific to it, and remaining length), while some of them have a variable header and/or payload (Figure 2).

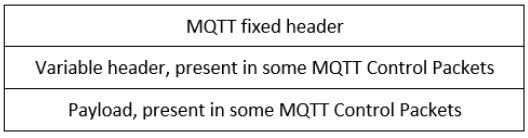

FIGURE 2. The general structure of the MQTT control packet.

TABLE 2. Control packets in MQTT.

Control packet	Description
CONNECT	Client requests a connection to a server
CONNACK	Server acknowledges the connection request to the client
PUBLISH	Message publishing
PUBACK	Publish acknowledgement
PUBREC	Publish received (QoS 2 publish received, part 1)
PUBREL	Publish release (QoS 2 publish received, part 2)
PUBCOMP	Publish complete (QoS 2 publish received, part 3)
SUBSCRIBE	Client subscribes to topics
SUBACK	Server acknowledges the subscription request to the client
UNSUBSCRIBE	Client unsubscribes from topics
UNSUBACK	Server acknowledges the unsubscription request to the client
PINSREQ	Client sends a PING request to the server
PINGRESP	Server sends a PING response to the client
DISCONNECT	Client sends a disconnect notification to the server for clear disconnection

The MQTT offers three Quality of Service (QoS) levels (in the PUBLISH packet) for message delivery from the publisher to the broker and from the broker to subscribers: at most once (QoS 0), at least once (QoS 1), and exactly once (QoS 2).

Below we describe in detail the PUBLISH and SUBSCRIBE control packets which roles are the most important from the perspective of this paper. The PUBLISH packet beside a fixed header, has a variable header which consists of a Topic Name, a Packet Identifier (present only for QoS 1 and QoS 2), and zero or more properties, followed by the Application Message (or update) as a payload. The SUBSCRIBE packet is composed of a fixed header, a variable header consists of Packet Identifier field and zero or more properties, and a payload containing a list of Topic Filters, each of which is followed by a Subscription Options byte.

Normally, when a client subscribes to a topic after the last update was sent to all subscribers, it must wait for the next update, to see, for example, the current status of a given sensor. However, the publisher can tell the broker to keep the last message on that topic by setting the RETAIN flag in the PUBLISH fixed header to 1. With the retained message, every new subscriber will see the last sent update (only one message can be retained per topic). The subscriber can also control receiving of retained messages by using two options in the Subscription Options byte which follows given topic filter. If the Retain As Published option is set to 1 then updates from the server keep the RETAIN flag they were published with. Next, Retain Handling option (2 bits) specifies whether retained messages will be sent on new subscriptions (0 or 1 to send a retained message with 1 only in a case when subscription currently does not exist, and 2 not to send it).

B. PUBLISH/SUBSCRIBE MODEL IN MQTT

As already mentioned, MQTT is a publish/subscribe protocol. Basically, this is a client/server model that allows the client to communicate with an endpoint. In this protocol two types of clients have been defined. The clients that send messages are called *publishers*. Other clients that receive the messages are called *subscribers*. These two types of clients never communicate directly with each other. In order to exchange messages, they utilize a central point that plays the role of a server and is called a *broker*. The role of the broker is to receive the messages sent by the publishers and to forward them to the subscribers. Publishers send messages on certain topics identified by their topic names. Subscribers use topic filters to subscribe/unsubscribe to/from specific topics by sending SUBSCRIBE/UNSUBSCRIBE control packets. When a broker receives the message, it determines the topic to which it needs to be sent and then transmits the message to those clients (subscribers) who have subscribed to that particular topic (Figure 3). Any message from the publisher to the broker, and from the broker to the subscribers is carried by the PUBLISH control packet. Note, that the publisher does not receive any information on how many subscribers received the published message.

The topic of a given message can be considered as a message subject. Basically, the topic name is an arbitrary UTF-8 encoded string. It can be also hierarchically structured using a forward slash (/) as a topic level separator. According to this rule, the topics have one or more

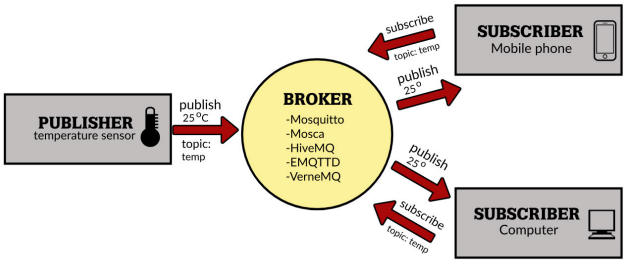

FIGURE 3. MQTT publish/subscribe model.

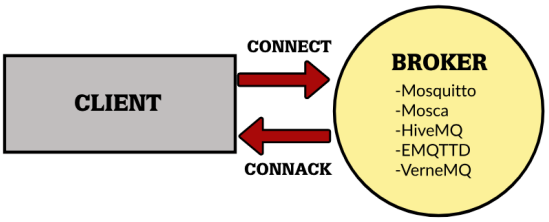

FIGURE 4. Establishing of the MQTT connection.

topic levels. Exemplary three level topic can take form as: `myFirm/office1/temperature`.

No prior initialization of the topic is required. It is not necessary to create a topic before publishing or subscribing to it. At least one character is needed to create a topic. The topic names are case-sensitive and can contain empty spaces. The forward slash alone is also a valid topic string. The subscribing clients can also use wildcards to subscribe to multiple topics. One can use one of two types of wildcards: single-level or multi-level. For creating a single level wildcard the plus symbol (+) is utilized. It replaces a single topic level. For example, in the topic `myFirm/+/temperature`, the plus symbol can be replaced by any string. According to above, these exemplary topics will match:

```
myFirm/office1/temperature
myFirm/lab1/temperature
```

The other type of wildcard is multi-level wildcard, created with the hash (#) character. For example, if one uses the topic `myFirm/office1/#` then the following exemplary topics will match:

```
myFirm/office1/temperature
myFirm/office1/lights/doorlight
```

The broker will forward to the client all messages which topic matches the subscribed one including the wildcard. In MQTT there are also themes that are intended for special purpose and they begin with the character `$`. Publishers cannot send messages to these topics. The most common topics of this kind are those that begin with `$SYS/`, which are used to display information that is specific to the broker.

C. MQTT CONNECTION

The MQTT protocol is TCP/IP based and the client and the broker need to have an implemented TCP/IP stack. The port 1883 is reserved for the transport of MQTT over TCP, while 8883 is reserved for the transport of MQTT over SSL/TLS.

The client sends a CONNECT control packet to the broker, to initiate a connection. The broker responds with a CONNACK control packet and a status code (Figure 4), and keeps the connection until the client sends a DISCONNECT packet or the connection breaks.

The session between the client and the broker can be non-persistent or persistent. A non-persistent (or clear) session is created by setting the Clear Session flag of the CONNECT control packet to 1. If the connection is lost or interrupted, all information about the client is lost on the broker side, and the client must re-subscribe to each topic again. A persistent session is created by setting the Clear Session flag to 0, and in this case, the broker needs to save the session state for the client all the time. When the client reconnects, the session state is available immediately. Persistent sessions are identified by the Client Identifier field only. After the client's disconnection, the broker must store all client subscriptions, and further or pending QoS 1 and QoS 2 messages that match any client subscriptions at the time of disconnection as a part of the session state.

The CONNACK control packet sent from the broker contains a *SessionPresent* flag, and if this flag is set to 1, it informs the client that there is a previous persistent session, which will be resumed. Additionally, if $SessionPresent = 1$, there is a zero return code in the CONNACK packet.

V. MQTT-BASED COVERT CHANNELS

In this section we present a comprehensive analysis of potential network covert channels that are applicable for the MQTT protocol. The covert channels are categorized by their hiding patterns and described using the *unified description method* proposed by Wendzel *et al.* [25]. Note, that the "application scenario" that is part of the unified description is always the same for these channels. Many different scenarios are possible, like general-purpose covert communication or data exfiltration from compromised systems, but the fundamental aspect is that Alice and Bob need to utilize MQTT to establish a policy-breaking communication and therefore use a covert channel.

A. SYSTEM MODEL

The investigated in this paper MQTT-based covert channel system model involves a covert sender (CS) and one or more covert receivers (CRs). Obviously, it is also possible to have multiple covert senders, but in our paper, we assume only one CS. Even more, as any client can be a publisher, the communication can be bidirectional, and the group of users can communicate together. CS and CRs can be on the same network or on different networks.

Moreover, we differentiate between two distinct submodels, Direct Covert Channels (DCC) and Indirect Covert Channels (ICC). In the system submodel DCC, CS directly communicates with CRs and in this case there are two possibilities:

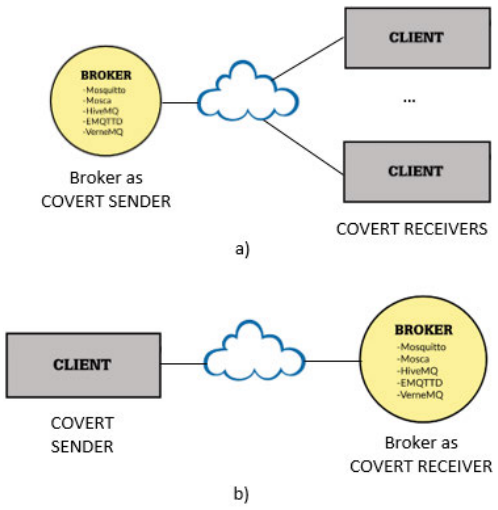

FIGURE 5. System submodel DCC: a) with the broker as the CS; b) with the broker as the CR.

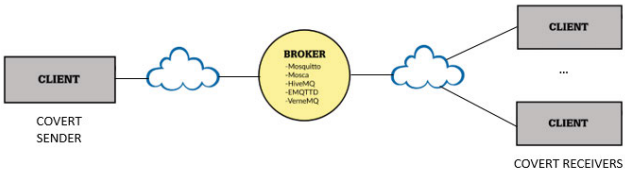

FIGURE 6. System submodel ICC (the broker is unaware of the hidden data exchange).

the broker can serve as the CS (DCCa) or the broker is the only CR (DCCb) – see Figure 5. In the system submodel ICC, CS indirectly communicates with the CRs through a broker as the intermediate node (Figure 6). This means that there is no direct interaction between CS and CRs and thus they do not have to be active at the same time (covert sending and covert receiving processes can be thus decoupled). In this scenario, the broker is unaware of the hidden data exchange and therefore it serves as a proxy node that forwards any messages that it receives.

According to the system submodels introduced above, the proposed covert channels can be divided into two groups: a group DCC where the direct well-known CCs slightly adjusted to the MQTT context are incorporated, and a group ICC in which indirect CCs that are MQTT specific have been placed.

Additionally, we assume that CS and CRs have well-synchronized clocks. Typically, the routers and wired devices are using Network Time Protocol (NTP) [26] for synchronization purposes and for the other networked devices different variants of NTP are usually utilized. For example, mobile phones utilize Simple Network Time Protocol (SNTP) [27] or Mobile NTP (MNTP) [28]. In case of IoT environments, SNTP or even the MQTT itself can be used for synchronization with some reference clocks. When MQTT is used, the broker or the cloud server publishes the current time to the client without taking into account one-way delays of the packets with the timestamp or any response message. In result, MQTT does not provide a precise synchronization that our CCs require. Mani *et al.* [29] suggest using the new Synchronization Protocol for IoT (SPoT) which maintains a clock accuracy of ca. 15ms at various noise levels. Thus, for the purposes of this work and presented CCs such a protocol should be utilized to ensure the required level of synchronization.

Finally, we assume that CRs know when the CS starts to send a new secret message. With respect to the attacker capabilities we assume that he can monitor, modify or fabric the traffic between the clients and the broker.

B. DIRECT COVERT CHANNELS DEPLOYED IN MQTT (DCC GROUP)

Several direct CCs can be deployed in MQTT for communication between the clients (publishers/subscribers) and the broker and they use well-known techniques slightly adjusted to the MQTT context. In particular, these CCs utilize modification of certain fields in the control packets. The text fields are encoded as UTF-8 strings, with the length not exceeding 65535 bytes and they are as follows:

1) **Application Message** in the payload of the PUBLISH packet. MQTT is data-agnostic, which means that it can carry virtually everything, e.g., images, audio, encrypted data, text in any encoding, etc. The CC based on this field can be also used to create an indirect covert communication between clients.
2) **Client Identifier** in the payload of the CONNECT packet (servers must allow lengths up to 23 UTF-8 encoded bytes).
3) **User Name** and **Password** fields in the payload of the CONNECT packet (each up to 65535 bytes).
4) 16-bit **Keep Alive** field in the CONNECT packet, which is the maximum time interval in seconds between two control packets send from the client.
5) 16-bit **Packet Identifier** (non-zero) in the PUBLISH packet when QoS $>$ 0 or in the SUBSCRIBE, UNSUBSCRIBE, SUBACK, UNSUBACK, PUBACK, PUBREC, PUBREL and PUBCOMP packets. For each new packet, the client and the server can assign Packet Identifiers independently of each other and the assigned value can be reused after processing the corresponding acknowledgement packet. Each acknowledgement packet has the same Packet Identifier as the packet that is acknowledged.
6) **Topic Name** in the PUBLISH packet (up to 65535 bytes).
7) **Topic Filters** in the payload of the SUBSCRIBE and UNSUBSCRIBE packets (up to 65535 B).

Prerequisites: Because the broker is a mandatory entity in the direct covert communication, the only prerequisites for DCC group of channels is other covert participants to be capable of exchanging messages with the broker.

Secret bits embedding and extraction: For the submodel DCCa, when broker as a CS wants to send some secret

message to the CRs, it needs to encode the message into the Application Message (DCC.1), Packet Identifier (DCC.5) or Topic Name (DCC.6). For the submodel DCCb, when some client as a CS wants to send a secret message to the broker, it needs to encode the message into the specific field, depending on the chosen CC.

Secret bits can be directly embedded into the fields Client Identifier, Password, and Packet Identifier, because they can have random values, and even into the Keep Alive or User Name fields, for similar reasons. Additionally, the same can be done into the Application Message field, because its content can be encrypted.

For the Topic Name and Topic Filters, besides the direct embedding of the secret bits, which is not so covert, the better way is to use one of the approaches listed below:

- modulation of the letter capitalization (e.g., lower case as binary 1, and upper case as binary 0),
- modulating the number of levels in the Topic Name (e.g., even number of levels denotes binary '1', while odd binary '0'),
- modulating the number of whitespace characters in the Topic Name (e.g., even number of characters embeds binary '1', while odd number binary '0'),
- utilization of different whitespace characters in the Topic Name (e.g., space character (U+0020) denotes binary '1', and non-breaking space character (U+00A0) binary '0'), etc.

Information hiding pattern: According to the pattern-based classification, the usage of User Name and Topic Name fields for hidden data purposes (DCC.1 and DCC.2) represents the *PS11. Value modulation* pattern, while exploitation of the fields Client Identifier, Password, Keep Alive and Packet Identifier belongs to the *PS10. Random value* pattern (DCC.3, DCC.4, and DCC.5, respectively). Finally, when the Application Message and Topic Filters from the packet payload are utilized for data hiding, the resulting covert techniques can be assigned to the *PS31. User-data value modulation & reserved/unused* pattern (DCC.6 and DCC.7).

C. INDIRECT MQTT-SPECIFIC COVERT CHANNELS (ICC GROUP)

1) COVERT CHANNEL USING TOPIC NAME AND TOPIC FILTERS FIELDS

The indirect covert channel (ICC.1) between a publisher (as a CS) and one or more subscribers (as a CRs) can be realized in the following way.

Prerequisites: At the beginning, all participants of the covert communication must have agreed upon some known first level of the Topic Name, for example, `Room`. Then CRs subscribe using multi-level wildcard (#) as a second level of the topic, for example, `Room/#` (Figure 7, step 1).

Secret bits embedding and extraction: Then, CS sends the secret message embedded in the rest of the Topic Name, by publishing updates with irrelevant (but believable) content for the newly created topic (Figure 7, step 2). This way all CRs obtain all updates for the topics that begin with `Room`

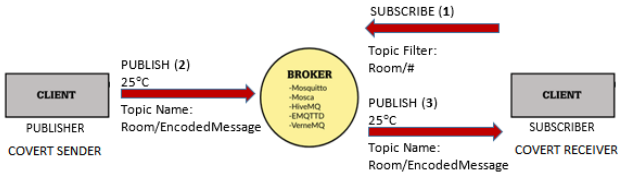

FIGURE 7. Indirect Covert Channel using Topic Name and Topic Filters fields.

(Figure 7, step 3), together with the hidden message that can be extracted. In such setup, the content of the updates is irrelevant, while in fact the names of these topics carry hidden messages.

Secret bits embedding in the Topic Name can be performed based on one of the potential approaches listed above, in the DCC embedding in Topic Name and Topic Filters.

Information hiding pattern: It must be noted that the covert channel described above represents the *PS11. Value modulation* pattern.

2) COVERT CHANNEL USING TOPIC ORDERING AND UPDATES PRESENCE/ABSENCE

Prerequisites: For this data hiding method (ICC.2) the covert sender and several covert receivers must first agree to use n topic names $T_1, \ldots, T_n$ and establish in advance the secret bits embedding scheme by topic ordering.

Secret bits embedding and extraction: All of the participants subscribe to previously agreed topics (Figure 8, step 1), which are enumerated and linked to secret bits. Then, if the covert sender wants to transmit a secret message, he publishes updates only to the topics that are mapped to bits which must be set to 1 in a certain message. For example, to transfer the 4-bit message 1011, the covert sender will publish status updates to the topics T_1, T_3 and T_4 (Figure 8, step 2), and all subscribers, together with the appropriate covert receivers, will receive these updates (Figure 8, step 3). From the presence/absence of the particular update on the agreed topic and previously agreed topic ordering scheme, covert receivers are then able to extract the hidden message.

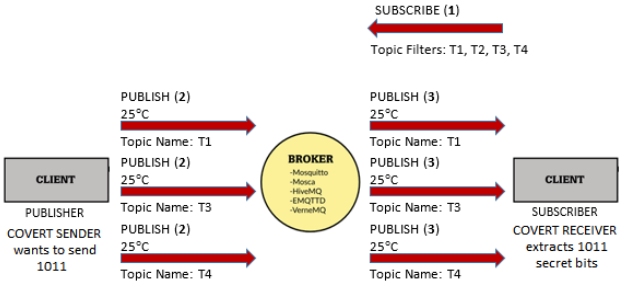

FIGURE 8. Indirect Covert Channel using Topic Ordering and Updates Presence/Absence.

Information hiding pattern: The core idea of this data hiding technique relies on pre-established topic names while these topics represent values in the protocol. The modulation

of their appearance makes this covert channel a form of the *PT11. Value Modulation* network information hiding pattern.

3) COVERT CHANNEL USING PERSISTENT SESSIONS

This one-bit unidirectional covert channel (ICC.3) utilizes the possibility of the client to create a persistent session with the server, where the server saves the client's session state all the time. Persistent sessions are identified by the Client Identifier field in the CONNECT control packet only.

Prerequisites: First, the covert sender provides its Client Identifier (for example senderID) to the covert receiver, and both of them will use it for creating connections with the server in different prearranged time slots.

Secret bits embedding and extraction: If the covert sender wants to send secret bit '1' to the covert receiver, then it will create a persistent session with the server, and for sending bit '0', it will create a non-persistent session with the server.

During the next time slot, the CS will disconnect itself, while the CR will create a persistent session with the server using the same senderID. The server will acknowledge obtained CONNECT packet with the CONNACK packet, in which the Session Present flag will be set to '1' (if there is a previous persistent session) or '0' (if there is no previous persistent session). After receiving and extracting the hidden bit in the Session Present flag, the CR will disconnect itself.

Note, that frequent connecting and disconnecting to the broker may not necessarily be considered as an anomaly as this is a typical practice if the client wants to save energy.

Information hiding pattern: Taking into consideration the hiding approach of this covert channel, it must be categorized as representing the *PS11. Value Modulation* pattern as one specific session (that is the actual *value* of the Value Modulation) is either created or not. Moreover, the CR polls the server that stores the secret data, which makes this channel an indirect one.

4) COVERT CHANNEL USING PRESENCE/ABSENCE OF THE RETAINED MESSAGE

If there is a stored retained message for the specific topic, when a new subscription that matches the topic name is established, the server will send this message to the subscriber. If the PUBLISH control packet is received by the server with RETAIN flag set to 1 and *non*-zero byte payload, the server will replace the retained message with the newly obtained one. When the PUBLISH control packet is received by the server with RETAIN flag set to 1 and zero byte payload, the server must remove the existing retained message and any new subscribers for that topic will not receive a retained message.

Prerequisites: For one-bit covert channel (ICC.4), the covert sender and the covert receiver first need to agree on the length of time slots that will be large enough so the CS will be able to transmit the hidden message and simultaneously CR can have enough time to send a new subscription and to obtain the retained message, if any.

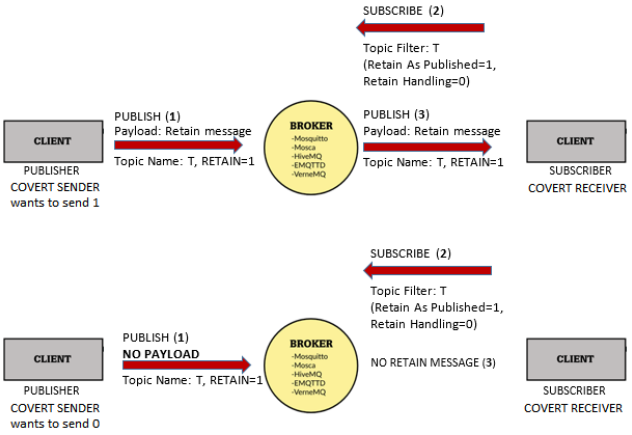

FIGURE 9. Indirect covert channel using Using Presence/Absence of the retained message.

Secret bits embedding and extraction: After that, the CS will send one secret bit per slot, as follows (Figure 9, step 1):

- The presence of a retained message can represent a binary '1'. For this purpose, the CS will send a PUBLISH control packet with RETAIN flag set to '1' and retained message as a payload.
- The absence of a retained message denotes a binary '0'. For this purpose, the CS will transmit the PUBLISH control packet with RETAIN flag set to '1' and zero byte payload.

Note, that the covert receiver needs to create a new subscription for the specific topic per time slot, by sending a SUBSCRIBE packet with the appropriate Topic Filter and subscription options: Retain As Published=1 and Retain Handling=0 (Figure 9, step 2). Also, it needs to use non-persistent (clear) sessions, to avoid receiving stored messages. On the server side, this action completely replaces the existing subscription with a new one. This way the covert receiver is able to establish if it is going to receive a retained message or not (in every time slot), which is identical to extraction of the secret bit '1' or '0' (Figure 9, step 3).

Information hiding pattern: The covert channel proposed above is a hybrid channel that combines *PS11. Value Modulation* and *PT2. Message Timing* patterns. In fact, it is the side effect (presence/absence of the retained message) that is visible to the covert receiver.

5) COVERT CHANNEL USING TOPIC ORDERING AND PRESENCE/ABSENCE OF THE RETAINED MESSAGES

Prerequisites: First, the covert sender and the covert receiver must agree to use n topic names $T_1, \ldots, T_n$ and their specific ordering. Then the ICC.5 is created as presented below.

Secret bits embedding and extraction: For sending n-bit long hidden message, the covert sender modulates the presence/absence of retained messages (as binary '1' or '0') in all these topics, by sending PUBLISH control packet with the RETAIN flag set to 1 and retained message as a payload / zero byte payload (as in the case of the previous covert channel).

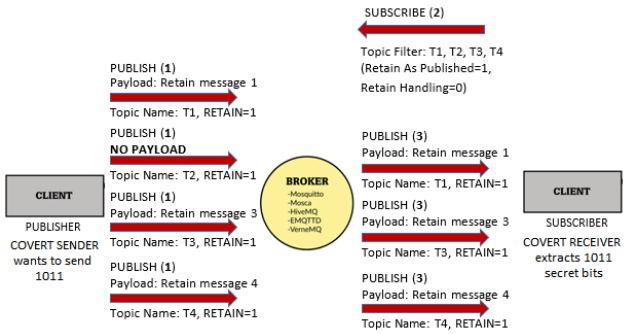

FIGURE 10. Indirect covert channel using topic ordering and Presence/Absence of the retained messages.

For example, for sending the 4-bit message 1011, the attacker will manipulate the presence/absence of retained messages in all the chosen topics, by setting and sending retained messages in the topics T_1, T_3 and T_4, and setting no retained message only for the topic T_2 (Figure 10, step 1). After that, he will disconnect.

The covert receiver makes new subscriptions to the topics T_1, T_2, T_3, T_4 (Figure 10, step 2), and receives the retained messages only for topics T_1, T_3 and T_4 (Figure 10, step 3). So, based on the prearranged scheme of topics ordering for data hiding purposes the CR is able to extract the secret message transmitted from the CS.

Information hiding pattern: This hybrid covert channel is a combination of the *PT11. Message Ordering*, *PS11. Value Modulation* and *PT2. Message Timing* patterns.

6) COVERT CHANNEL USING INFORMATION SPECIFIC TO THE BROKER

Topics that begin with `$SYS/` are used to display information that is specific to the broker and publishers cannot send updates on them. At the moment, there is no official standardization of such topics. However, they can be exploited for covert communication purposes, too.

If a covert sender has a possibility to control connecting and disconnecting of certain clients to the server, this can be used as a covert communication channel (ICC.6) with some covert receivers connected to the same server. This can represent an exemplary scenario where CS is able to control several sensors in a building.

Prerequisites: The covert sender and the covert receiver need first to investigate the dynamics of connections/disconnections of different clients. The dynamics of the clients' connections can be investigated by reading the number of currently connected (NC) clients on the broker over the chosen time period, which can be done by subscribing the topic:

`$SYS/broker/clients/connected`

On the other hand, the dynamics of disconnections can be investigated by reading the number of persistent (with clean session disabled) clients (ND) that are registered at the broker but are currently disconnected, over the chosen time period, which can be done by subscribing to the topic:

`$SYS/broker/clients/disconnected`

We will explain the proposed covert channel inner workings by using NC values (however analogously ND values can be used). Both clandestine communication sides need a pre-phase where they will collect information over some period of time about historical NC values and after that they will start the hidden data transfer. Even more, during their secret communication they will continue to query NC values.

Secret bits embedding and extraction: CS and CR need to agree on small triples of time slots $(t_{i1}, t_{i2}, t_{i3}), i = 1, 2, \ldots$, in which the following steps will be undertaken:

1. In the time slot t_{i1}, CS and CR will obtain the current *NC* value and will calculate an Average Number of Connected (ANC) clients and Standard Deviation of Connected (SDC) clients for the server, and the following value:

$$\mathrm{DT} = \begin{cases} \mathrm{SDC} - (\mathrm{NC} - \mathrm{ANC}), & \text{if NC} > \mathrm{ANC} \\ \mathrm{SDC} - (\mathrm{ANC} - \mathrm{NC}), & \text{otherwise} \end{cases}$$

2. Sending secret bit in the time slot t_{i2}:
- For sending binary 1, if $NC > ANC$ the covert sender will connect *DT* clients, otherwise it will disconnect *DT* clients;
- For sending binary 0, if $NC > ANC$ the covert sender will disconnect $SDC - DT$ clients, otherwise it will connect $SDC - DT$ clients.

Note, that *DT* can be also equal to zero which means that in this case, when sending secret bit '1' the CS should not do anything, and still the secret bit will be transmitted.

3. Receiving secret bit in the time slot t_{i3}. The CR will obtain the *NC* value for the time slot t_{i3}, and if this value is around the average number of connected clients, *ANC*, it will interpret it as binary 0. Otherwise, if it is around minimal value, $ANC - SDC$ or maximal value, $ANC + SDC$, it will extract binary 1.

As already mentioned, similar channel can be established using Average Number of Disconnected (AND) persistent clients, Standard Deviation of Disconnected (SDD) persistent clients, and Number of Disconnected (ND) persistent clients. In this case the procedure is as follows:

1. During the time slot t_{i1}, CS and CR will obtain the current *ND* value and will calculate the Average Number of Disconnected (AND) persistent clients and Standard Deviation of Disconnected (SDD) persistent clients for the server, and the following value

$$\mathrm{DT} = \begin{cases} \mathrm{SDD} - (\mathrm{ND} - \mathrm{AND}), & \text{if ND} > \mathrm{AND} \\ \mathrm{SDD} - (\mathrm{AND} - \mathrm{ND}), & \text{otherwise} \end{cases}$$

2. Sending secret bit within the time slot t_{i2}:
- For sending binary 1, if $ND > AND$ the covert sender will disconnect *DT* clients, otherwise it will connect *DT* clients;
- For sending binary 0, if $ND > AND$ the covert sender will connect $SDD - DT$ clients, otherwise it will disconnect $SDD - DT$ clients.

3. Receiving secret bit in the time slot t_{i3}. The receiver will obtain the *ND* value for the time slot t_{i3}, and if this value is around the average number of disconnected persistent clients, *AND*, it will interpret it as binary 0. Otherwise, if it is around minimal value, $AND - SDD$ or maximal value, $AND + SDD$, it will extract binary 1.

One can control the clients' connecting and disconnecting using several means that deploy some kind of clients' resources management. For example, possible solutions include turning ON/OFF the access to the internet or the clients' energy supply.

Information hiding pattern: The necessity of synchronized timing between CS and CR does not make this channel a timing channel (all indirect channels require a somehow synchronized timing, also, e.g., the channel proposed in [20]). Instead, the value of the (disconnected) clients NC (ND) and the related values are manipulated, making this channel a special variant of the *PS11. Value Modulation* pattern. However, due to its novel nature, one can see this channel as a new child pattern of PS11, which we refer to as *PS11c. Value Influencing Pattern*. Moreover, as CS can be distributed here, this channel can be considered a distributed covert channel (see [22] for details on this).

VI. PROPOSED COVERT CHANNELS' PROPERTIES

When MQTT is used in practice, there are two types of publishers. The first type publishes messages in predetermined intervals, which can vary in milliseconds and seconds. For example, the sensor monitoring pressure or brightness that are constantly changing, need to scan and report quite frequently (e.g., every second). The second type publishes messages only on the state changes, as when you have a sensor monitoring a given window or a door (e.g., is it open or closed?). The window or the door can remain closed all day, but if it is open, this status is needed to be reported in seconds. Thus, this sensor needs to scan continuously, but reports only on status changes.

We are going to investigate more properties of the network covert channels presented in this paper, and particularly, for those that belong to the ICC group (indirect covert channels) as they are MQTT-specific and their detection/elimination is more challenging that those from group DCC (which are typical network storage covert channels). Frequency of the state change in the second type of publishers is different for different sort of clients (e.g., for temperature sensor or actuator responsible for door) and usually depends on the environment where they are used (e.g., the state ''open door'' occurs with different frequency at the airport, supermarket or at the office), so we decided not to work with this type of publishers. For that purpose, we will make the following assumption: we will use only publishers of the first type, i.e., those that publish in the predetermined time intervals.

In the remaining of this section we will discuss the three most important performance metrics for each covert channel, i.e, bandwidth, undetectability, and robustness.

A. COVERT CHANNEL BANDWIDTHS

The bandwidth as a covert channel property determines how much secret bits are transferred per second. Below we present the detailed analysis of the potential bandwidths of the proposed covert channels.

For DCC.1 (DCC.3), because Application Message (User Name and/or Password) can be used directly as a hidden message, one can hide maximum up to $P = 65535B = 524280$ ($P = 2 \times 65535B = 1048560$) bits per control packet. If the time interval between two message updates (two CONNECT packets) is t seconds, the resulting bandwidth will be P/t bps.

Similarly, for DCC.2 (DCC.4), the maximal size of Client Identifier (Keep Alive), and by this, the maximal size of hidden bits per CONNECT packet is $P = 23B = 184$ ($P = 16$) bits. If the time interval between two CONNECT packets is t seconds then the resulting bandwidth will be P/t bps.

The maximal number of hidden bits per control packet for the DCC.5 is $P = 16$ bits, so if the time interval between two control packets with Packet Identifier field in them is t seconds then the bandwidth will be P/t bps.

For the DCC.6, DCC.7 and ICC.1 the resulting bandwidth depends on the variation that will be used. The topic name or topic filters themselves can be hidden messages, with a maximal size of up to $P = 65535B = 524280$ bits per message, but this is not recommended because anybody sniffing the network could then read the message. Also, if topic name or topic filters are encrypted hidden messages, this can be also be detected, because in normal applications, topic names and filters are some known words, as temperature, light, door, sensor, etc. or a group of words as sensor one, kitchen door, etc. A better way is to encode the message into two different whitespace characters or alternatively in the even\odd number of whitespace characters or in the even\odd number of levels in the Topic Name or Topic Filters, and in this way, $P = 1$ bit per message will be send. In all of these cases, created CCs will be challenging to detect, because only two predefined topics can be used in each case, and sending hidden bits will be equal to publishing consecutive messages in one or in the other topic (DCC.6, ICC.1), or consecutive subscribing to one or to the other topic (DCC.7). If the time interval is t seconds, the resulting bandwidth will be P/t bps.

For ICC.2, if there are n prearranged topics between the CS and CRs, and if the CS publishes or not randomly in each topic every t seconds, then the potential bandwidth will be n/t bps. This is possible because the broker immediately after receiving the update for a given topic sends it to all subscribers of that topic.

For ICC.3, let the prearranged alternating time interval be t seconds. This means in two consecutive intervals, each of t seconds, first, the CS will send 1 bit by connecting to the broker with the predefined senderID, and then disconnecting, and second, the CR will receive one bit by connecting to the broker with the same senderID,

and then disconnecting. So, the resulting bandwidth will be $1/(2t)$ bps.

For ICC.4 and ICC.5, we can use two consecutive intervals, each of t seconds, first, for CS to publish the message updates and the server to replace or remove the retain messages per topic, and second, for CR to subscribe to the given topics. If we have n prearranged topics between the CS and CR, the resulting bandwidth for ICC.5 will be $n/(2t)$ bps. The ICC.4 is only a special case of the ICC.5 for $n = 1$, so, its resulting bandwidth will be $1/(2t)$ bps. The CCs that use retain messages, are especially suitable for use when the publishers are configured to publish on state changes. In this way, each new subscriber will receive the last reading of the device.

The most difficult to estimate is the potential bandwidth of the ICC.6. As already presented in previous section three consecutive intervals (t_1, t_2, t_3) are used for data hiding purposes. The t_2 interval should be long enough to connect or disconnect *DT* clients. In order to empirically establish these intervals we have performed more than 20 consecutive connections and disconnections to the *test.mosquitto.org* broker, and the average time between sending a CONNECT control packet to the broker and receiving the CONNACK packet from the broker was 0.25ms with a standard deviation of 0.14ms. If the IoT device is turned off, then this time will be extended with the time needed for the device to turn on. The time elapsed from the moment when the client sends DISCONNECT notification (or the connection is terminated abnormally) and the moment when the broker updates its database with new information, is more difficult to estimate. Because the transport protocol is TCP, this time will depend on the time needed to ''gracefully'' finish (or abnormally finish) the TCP session. When these three values are estimated, the resulting bandwidth will be $1/(t_1 + t_2 + t_3)$ bps. For this channel, we queried the values *NC* and *ND* from the *test.mosquitto.org* broker every minute for 24 hours, starting from 22:30 at 02.09.2019 till 22:30 at 03.09.2019. The obtained results *ANC*, *SDC*, *AND* and *SDD* are provided in Table 3. Based on the results from this table it can be seen that the variation in *SDD* values is much smaller than *SDC* values, and they are in range [5.23, 16.39]. In fact, *SDD* values are between 5 and 10 (except 2 peaks), which correspond to smaller number of devices that needs to be manipulated for ICC.6 CC. Also, this means that the second variation of ICC.6 CC with *ND* values is better to use in practical scenarios.

TABLE 3. Real-life data from test.mosquitto.org from 22:30 at 02.09.2019 till 22:30 at 03.09.2019.

Time	ANC	SDC	AND	SDD
22:30-23:30	1288.80	8.19	702.47	7.28
23:30-00:30	1288.83	8.94	715.73	5.80
00:30-01:30	1290,58	7.96	727.30	5.61
01:30-02:30	1296.73	6.83	738.42	6.04
02:30-03:30	1292.50	10.04	748.53	6.58
03:30-04:30	1296.18	10.21	761.43	7.28
04:30-05:30	1327.35	17.53	778.38	6.20
05:30-06:30	1132.05	149.86	792.02	5.53
06:30-07:30	1023.87	20.22	803.70	5.36
07:30-08:30	1077.72	13.33	817.57	7.67
08:30-09:30	1105.28	19.23	839.45	8.90
09:30-10:30	1172.53	24.06	855.05	5.23
10:30-11:30	1198.45	10.35	869.20	7.11
11:30-12:30	1188.12	16.81	887,00	6.18
12:30-13:30	1171.10	8.89	914.25	11.33
13:30-14:30	1172.73	6.33	936.82	7.57
14:30-15:30	1175.18	8.56	968.67	16.39
15:30-16:30	1178.98	8.90	1008.75	8.14
16:30-17:30	1165.02	10.77	1025.43	7.15
17:30-18:30	1118.25	17.22	1040.53	7.22
18:30-19:30	1106.47	9.84	1051.57	9.29
19:30-20:30	1040.95	82.67	1071.12	8.73
20:30-21:30	1076.13	56.70	1084.77	8.11
21:30-22:30	1114.53	8.36	1096.58	7.28
Avg.	**1179.10**	**22.58**	**884.78**	**7.58**

B. UNDETECTABILITY

Another important covert channel property is undetectability, which is inability of some third party (an attacker, a warden) to detect a hidden message in a given carrier or in another words, the inability to distinguish clandestine traffic from a legitimate one. Usually, third parties can use the analysis of some statistical properties of the captured traffic's data and compare obtained results with the one from legitimate traffic recordings. Different data mining techniques can be used for this purpose. However, below we decided to review potential countermeasures that can be used to thwart the proposed CCs.

Often, the Application Message has some structure which depends on the type of the IoT device that publishes it. For example, a temperature sensor sends only temperature readings, and a humidity sensor transmits humidity readings. So, one possible countermeasure against the DCC.1 CC is to look for content in the Application Message which is unusual for the specific IoT device (e.g., some unusual text for the temperature sensor).

One possible detection method for DCC.2 and DCC.3 is to monitor Client Identifier, User Name and Password fields in the CONNECT control packets and to look for many different values from the same IP addresses or the same MAC addresses.

DCC.4 can be possibly detected by monitoring the Keep Alive field in the consecutive CONNECT control packets from the same IP addresses or the same MAC address while tracking keep alive values. The occurrence of several different values could indicate the presence of a covert channel.

The DCC.5 would be the most difficult to detect because it is expected that each new packet has a new random value in the Packet Identifier field. One possibility to discover covert communication is to look at the acknowledgment packets that need to have the same value of the Packet Identifier field as in the packet that is acknowledged (e.g., PUBACK: Packet Identifier should be the same as in the PUBLISH packet that acknowledges it).

One possible countermeasure for detection of the DCC.6, DCC.7 and ICC.1 CCs, is to inspect topic names in published messages (or topic filters in SUBSCRIBE and

UNSUBSCRIBE packets for DCC.7), and to look for unusual behavior, like publishing in (or subscribing to/unsubscribing from) many different topics, in topics with long names or names with random characters, in topics with strange names where different letter capitalization or different whitespace characters are used, etc.

The topic ordering exploited by ICC.2 and ICC.5 could potentially be detected by comparing the topic order of a given flow with legitimate topic ordering, e.g., by inspecting the entropy.

A possible countermeasure for detection of the ICC.3, is to keep a record of (Client Identifier, MAC address) pairs, and to look for at least two pairs with the same Client Identifier field and different MAC addresses. Alternatively, one needs to inspect the log file, and search for several CONNECT control packets with the same Client Identifier field and different IP or MAC addresses, in relatively short time intervals.

ICC.4 and ICC.5 can be detected by counting the number of SUBSCRIBE control packets per client, and the number of PUBLISH control packets without payload per client. If the broker administrator has some kind of statistics of these numbers in the legitimate traffic, these CCs will be detected if there is a large difference when compared to legitimate values, especially in the case where discrepancies occur together.

Finally, the countermeasure for detection of the ICC.6 can be based on counting the numbers of CONNECT and DISCONNECT control packets per client on the broker side. If there are large differences in comparison to legitimate ones, especially on the same subset this could be a hint of potential existence of covert communication.

C. ROBUSTNESS

The robustness of each of the suggested indirect CCs that use PUBLISH control packets on the CS side (i.e., ICC.1, ICC.2, ICC.4, ICC.5) can be negatively affected if someone else, besides the publisher, publishes in the same topic(s), where the specific CC is deployed. This is possible, because the broker sends a specific topic update to all subscribers, without the information of the publisher in the control packet. One way to retain the robustness of these CCs, is to use the authorization policies for clients in the MQTT broker in order to limit their ability to publish messages in a specific topic. This can be done by implementing topic permissions on the broker side, that specifies which client can publish in a specific topic. The MQTT 3.1.1 specification allows clients' disconnections only in the case when an unauthorized publish occurs. Still, because the identification of the client is done only by its Client Identifier (ClientId), and thus everyone who knows it, can publish in this topic. It is interesting that if someone attempts to connect to the broker with the same ClientId as an existing connected client, then the existing client connection is dropped and the legitimate client is disconnected. So, this way, one can perform a Denial of Service (DoS) attack to a specific client, together with an identity theft, while using it for the needs of the CC.

The other two indirect CCs (ICC.3, ICC.6) heavily depends on the number of connecting and disconnecting clients. So, in the environments, where clients need to save their energy, and because of that, normally they often connect and disconnect from the broker, these two CCs cannot remain robust. So, when disconnections occur only on request of the client for purpose of the given CC, robustness of the CC is increased. Here also, one can influence CC robustness, if by some means, he can force the broker to disconnect the client. As previously mentioned, it can be achieved by sending CONNECT control packet with the same ClientId as the connected client.

Direct CCs are more robust, especially if the CS and CRs are on the same network. On different networks, an attacker performing Man-in-the-Middle (MitM) attack can modify or fabricate covert control packets. One can also impersonate itself as a broker and send modified or fabricated messages to the other covert parties.

VII. EXPERIMENTAL EVALUATION OF THE CHOSEN COVERT CHANNEL

In order to prove that the proposed covert channels are feasible and effective we have created an ICC.2 prototype and we performed its experimental evaluation.

For our experiments we used realistic but self-generated MQTT traffic because we were not able to find any useful, publicly available datasets. We created one EC2 instance on AWS[7] for the Mosquitto[8] broker v.1.4.8. As a publisher (or a covert sender) we implemented an MQTT client using the Java Paho Client library[9] v.0.4.0. As the subscriber (or a covert receiver) we used the MQTT.fx[10] v.1.7.1 client program (Figure 11). To record the traffic, we used Wireshark[11] v.2.6.8.

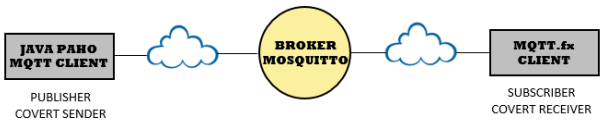

FIGURE 11. Experimental test-bed.

We investigated three different variants of the ICC.2, i.e., where:

- *Variant 1:* 2 topics (T1 and T2),
- *Variant 2:* 3 topics (T1, T2, and T3),
- *Variant 3:* 4 topics (T1, T2, T3, and T4).

are utilized for data hiding purposes.

First, we recorded 100 traffic samples for each version of the legitimate traffic (i.e., without covert channel). For the legitimate traffic we have a situation where the publisher publishes message updates in 2, 3 or 4 topics, depending on

[7]https://aws.amazon.com/
[8]https://mosquitto.org/
[9]https://www.eclipse.org/paho/clients/java/
[10]https://mqttfx.jensd.de/
[11]https://www.wireshark.org/

TABLE 4. Summary of the introduced covert channels.

Covert Channel	System sub−model	Pattern	Bandwidth	Description
DCC.1	all	PS11	up to 524280/t bps	Direct or indirect CC by using PUBLISH: Application Message
DCC.2	DCCb	PS11	up to 184/t bps	Direct CC by using CONNECT: Client Identifier
DCC.3	DCCb	PS10	up to 1048560/t bps	Direct CC by using CONNECT: User Name and/or CONNECT: Password
DCC.4	DCCb	PS10	up to 16/t bps	Direct CC by using CONNECT: Keep Alive
DCC.5	DCCa, DCCb	PS10	up to 16/t bps	Direct CC by using Packet Identifier in 11 Control Packets
DCC.6	DCCa, DCCb	PS31	up to 524280/t bps	Direct CC by using PUBLISH: Topic Name
DCC.7	DCCb	PS31	up to 524280/t bps	Direct CC by using SUBSCRIBE: Topic Filters or UNSUBSCRIBE: Topic Filters
ICC.1	ICC	PS11	up to 524280/t bps	Indirect CC using PUBLISH: Topic Name and SUBSCRIBE: Topic Filters
ICC.2	ICC	PT11	n (topics)/t bps	Indirect CC using topic ordering and updates presence/absence
ICC.3	ICC	PS11	1/(2t) bps	Indirect CC using persistent sessions
ICC.4	ICC	PS11+PT2	1/(2t) bps	Indirect CC using presence/absence of the Retained message
ICC.5	ICC	PT11+PS11+PT2	n (topics)/(2t) bps	Indirect CC using topic ordering and presence/absence of the Retained messages
ICC.6	ICC	PS11.c	1/t bps	Indirect CC using information specific to the broker

the ICC.2 variant, by randomly choosing in every second to publish/not to publish for each topic separately. Also, the time between each publishing event is random (in milliseconds). Each traffic sample lasted 10 minutes.

After that, we created the covert channel for each of above-mentioned variants of the ICC.2 and we sent secret messages in an encrypted form (ciphered with the AES), again with traffic samples that lasted 10 minutes. The covert channels were implemented using the Java programming language.

For the ICC.2 using 4 topics (Variant 3) we have performed another experiment. We sent secret messages in clear ASCII, just to see how this influences the detectability compared with the encrypted traffic. Similarly like in previous cases we used 100 traffic samples that lasted 10 minutes.

A. BANDWIDTH

In order to establish the achievable bandwidth of the ICC.2, we experimented using all its three variants, i.e., where 2, 3 or 4 topics were utilized for data hiding purposes. Because we use a time interval of one second for publishing/not publishing in the 2, 3 or 4 topics then, obviously, the resulting experimental bandwidth is 2 bps for 2 topics, 3 bps for 3 topics, and 4 bps for 4 topics. In each case, every second sending the binary 1 corresponds to publishing in the specific topic, while, sending the binary 0 corresponds to not publishing in the specific topic.

Figure 12 represents the first 100 seconds of one of the 100 performed experiments for ICC.2 with 4 topics. More precisely, this figure represents transmission of 400 secret bits in 100 seconds from the 7th till the 107th second, also showing the points in time when publishing in T1-T4 occurs. The first 7 seconds are for connecting and synchronizing. It must be also noted that for other ICC.2 variants the obtained results were analogous.

B. UNDETECTABILITY

To exemplify a detectability approach, we investigate the undetectability of ICC.2 CC. In a first experimental attempt,

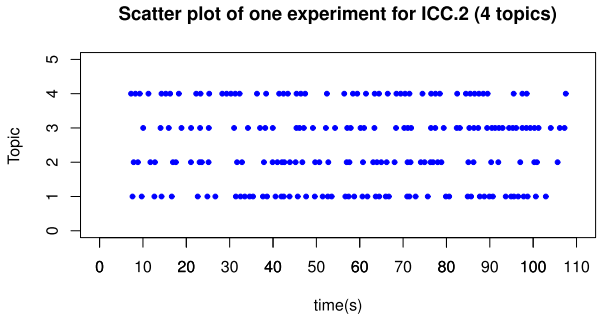

FIGURE 12. ICC.2 with 4 topics, sending 400 secret bits in 100 seconds.

we tried to detect the ICC.2 based on data-mining by monitoring the order of topics in network traffic. We generated 100 CC flows for the ICC.2 and compared its appearance of topics with 100 legitimate flows. First, we recorded all topics in the order of their appearance in a string S. For instance, if a flow would contain "Publish Message [T1]", followed by "Publish Message [T3]", "Publish Message [T4]" and "Publish Message [T1]", then our resulting string S would be "1341". Next, we compressed the string using a compressor $\Im$, i.e., $C = \Im(S)$ and compared its original length to the compressed length, i.e., S/C, resulting in a compressibility score κ. We determined the optimal string length for S by testing a set of parameters between 500 and 2,000 topics. In result, over all the scenarios that we test in the remainder of this section, the string length of 1,100 was a good choice.

The detection approach based on the string compressibility is based on a covert timing channel detection method by Cabuk *et al.* [30] but could not be applied directly to our covert channel as we took topic order (instead of inter-arrival times) as an input, generated a different string and needed to find other window sizes and thresholds for κ. This adjustment of a covert channel countermeasure is known as *countermeasure variation* [31]. Finally, we tried to define an optimal combination of string size and κ-threshold to classify between legitimate and CC traffic.

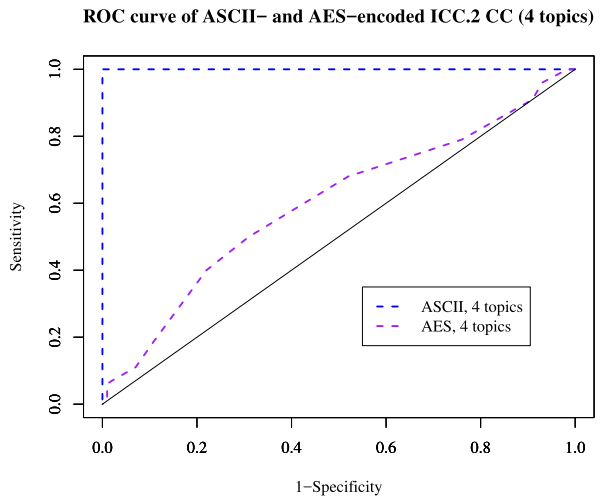

FIGURE 13. Detectability of a ICC.2 variant 3 (using 4 topics) with ASCII vs. AES-encrpted content.

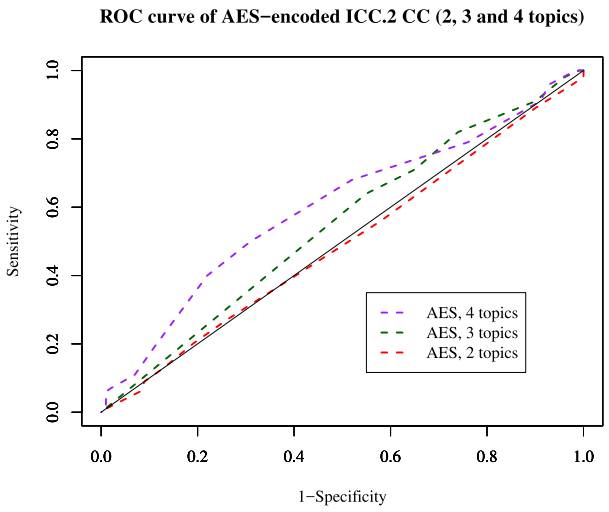

FIGURE 14. Detectability of ICC.2 that uses AES-encryption in combination with 2, 3 and 4 topics.

When our channel encodes secret data in trivial ASCII traffic format using 4 topics (i.e., the presence of 4 topics represents 4 bits of the ASCII characters),[12] the detection was providing excellent results (100% F-Score).

While such a channel might be the most suitable and most easy to implement by an attacker, one could also imagine that an attacker could encrypt the traffic in advance using AES. In the case of AES-encrypted traffic that is also encoded in 4 topics, the detectability decreases clearly (66.9% F-Score). For comparison, the ROC curves of both variants of the ICC.2 CC are shown in Fig. 13. The ASCII channel's AUC (Area Under Curve) was 0.995 while the AES channel's was 0.5996. Note, that we generated 100 legitimate and 100 covert flows for every scenario.

Next, we wanted to examine whether an attacker could render the ICC.2 CC harder to detect if a different number of topics is used together with the AES-encrypted content. For this reason, we generated 100 legitimate and 100 covert channel-based flows for scenarios where 2, 3 and 4 topics are utilized. It turned out that in this case the detectability *slightly* decreases when fewer topics are used. The maximum F-Score for 2 topics was 66.66% while it was 66.89% for 3 topics, and 66.89% for 4 topics. The differences are also visible in the ROC curves (Fig. 14). Overall, the channel using only 2 topics could not be detected better than guessing; the accuracy was 50.5%, the precision 0.51% and the AUC was 0.4933. For the channel with 3 topics, accuracy increased to 54.5% and precision to 53.78%, which is still hardly detectable, while resulting in an AUC of 0.5472. Depending on the used κ threshold, the channel with 4 topics could be detected either with 52.5% accuracy (but 85.71% precision) or with 59.5% accuracy (61.73% precision).

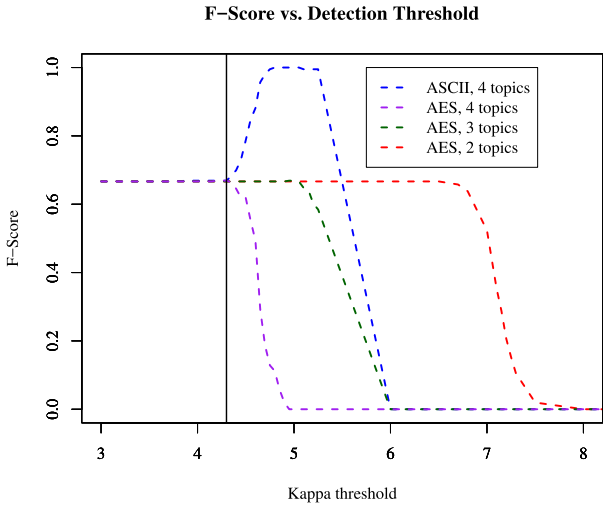

FIGURE 15. Optimization problem for determination of the optimal threshold – 4.3 performs best, overall, but higher thresholds benefit specific other channels.

Afterwards we analyzed the optimal detection thresholds for κ (Fig. 15) under consideration of maximizing the F-Score. The optimal thresholds were similar in case of the AES-encoded channels. For 2 topics, the best performing values were between $\kappa = 3.0$ and $\kappa = 6.5$ (all providing the same F-Score of 66.66%). For 3 topics, the best performing values were between $\kappa = 3.0$ and $\kappa = 5.15$ (optimal: $\kappa = 4.95$ and $\kappa = 5.0$) and for 4 topics, best values were between $\kappa = 3.0$ and $\kappa = 4.4$ (optimal: 4.3). Overall, the threshold $\kappa = 4.3$ provided the best *average* F-Scores to detect all three AES channels (all F-Scores between 66.66% and 66.89%). However, the ASCII channels' best

[12]Our channel operates in 1-second intervals. This means that within one second, the *appearance* of the particular topics in publishing messages encodes that the related bits are set to "1", otherwise they are considered as "0". For example, if during the first interval, the topics T3, T4 but not T1 and T2, would appear, then the first four hidden bits would be "0011". If, during the next interval, the topics T2 and T3 would appear, the next four hidden bits would be "0110". Thus, two seconds form one ASCII byte (in this example case, it would be the byte "00110110", i.e., "6" in ASCII).

detectability was given for thresholds between 4.8 and 5.25 (all close to or exactly 100% F-Score), while $\kappa = 4.3$ would only provide an F-Score of 66.89%, making it necessary to apply two separate thresholds for κ if both, the ASCII channel and the AES channels must be detected.

Finally, when realistic conditions are expected, then only a fraction of the flows might contain an ICC.2 covert channel. For this reason, thresholds should be selected in a way that would keep the false-positive-rate (FPR) at a minimum. As shown in Fig. 16, the FPR for the ASCII and the AES channel with 4 topics drops close to zero at $\kappa = 4.75$, rendering this threshold suitable for realistic conditions where the number of false-alarms must be minimized. The F-Score for the ASCII channel with 4 topics and $\kappa = 4.75$ is 99.5% while the FPR is 1%, the F-Score even increases to 100% with an FPR of 0% for $\kappa = 4.8$.

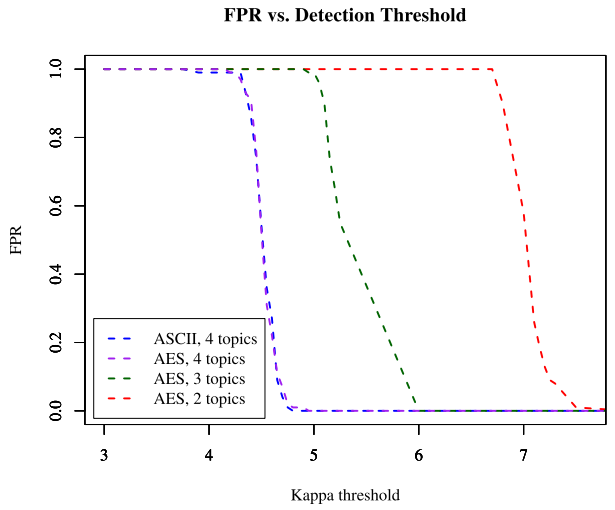

FIGURE 16. False-positive-rate of ICC.2, depending on encoding and number of utilized topics.

However, for the AES channel with 4 topics, the F-Score for the same threshold of 4.75 is only 12.84%. There is no threshold available to detect the AES 4-topic channel with an F-Score of at least 50% while maintaining FPR below 30%, rendering this channel not detectable under conditions where the fraction of covert channel traffic is low. The same applies to the AES channel with 3 and 2 topics (but with worse results). We can thus conclude that only the ASCII channel can be detected under condition where the fraction of covert channel traffic is either high or low while the AES-based channels are not detectable under realistic conditions.

We expect malware authors to implement simple versions of a covert channel first (as can be seen in the cases of currently known stegomalware that apply usually trivial methods) but it would be enough to encrypt the hidden content to render ICC.2 CC undetectable when solely the compressibility of topic appearances is analyzed. Thus, we conclude that this countermeasure variation solely works for the ASCII version of ICC.2 CC.

C. ROBUSTNESS

In order to establish how robust the ICC.2 covert channel is we investigated how it behaves under the influence of increased delays and packet losses, separately. This intended to simulate real-life networks conditions.

TABLE 5. The number of flipped bits in 120 transferred secret bits due to network delays.

	10ms		50ms		100ms	
	0 to 1	**1 to 0**	**0 to 1**	**1 to 0**	**0 to 1**	**1 to 0**
Exp 1	1	1	1	2	4	7
Exp 2	0	1	1	2	4	6
Exp 3	1	5	2	6	3	7
Avg.	**0.67**	**2.33**	**1.33**	**3.33**	**3.67**	**6.67**

Table 5 shows the number of flipped bits (the number of bits received with errors) on the receiver side. The measurement is done on the bit stream of length 120 secret bits, with three different introduced network delays (of 10ms, 50ms and 100ms), and with three experiments. Separately, the numbers of flipped zeros and the number of flipped ones are provided. Transmission errors in this case happen because some of the delayed message updates are published in the next second, instead in the current second. This way, 1 or 2 bits can be flipped. For example, if the message update in T4 which corresponds to binary 1 in the current second, is delayed for the next second, then in the current second 0 bit will be received (first flipping). If in the next second, the bit corresponding to publishing/not publishing in T4 is 0, then delayed T4 message update will flip it to 1 (second flipping), but if it is 1, the bit will remain the same.

TABLE 6. The number of flipped bits in 120 transferred secret bits due to packet losses.

	No. of 1	**No. of 0**	**1%**	**5%**	**10%**	**15%**
Exp 1	65	55	0.65	3.25	6.50	9.75
Exp 2	60	60	0.60	3	6	9
Exp 3	69	51	0.69	3.45	6.90	10.35

On average, our results indicate that for 10ms of introduced delay, 2.5% of the bits are received with errors, for 50ms 3.9% are received with errors, and for 100ms 8.6% of the bits are received with errors. Therefore, one can observe that with increasing delays the number of errors increases.

MQTT within the transport layer is using TCP. If the clients are always connected even with QoS 0, there will be no packet losses, because TCP is a reliable protocol. If the client is not always connected, with QoS 0, packets will be lost during the time when client is offline or not connected to the broker. For the ICC.2 this means that only a subset of ones will be flipped to zeros, while original zero bits will remain unchanged. So, in a case of 120 transferred secret bits, for Experiment 1 with 65 ones, for 1% of the packets lost, we will have 0.65 bits interpreted as zeros, for 5% 3.25 bits will be with errors, for 10% 6.50 bits and for 15% 9.75 bits will be with errors (Figure 6). Therefore, one can observe that with

increasing packet losses the number of flipped bits elevates, too.

VIII. CONCLUSION AND FUTURE WORK

In this paper, we have shown that MQTT-based covert channels are feasible. We introduced seven direct and six indirect covert channels and discussed for them possible countermeasures. Performed experimental evaluation for one of the channels revealed that the detection of such channels could be a challenging task. We plan to further analyze the detectability of all presented covert channels in the future work.

To the authors' best knowledge, these are the first covert channels proposed for the publish−subscribe model. Covert channels ICC.1 and ICC.2 are quite generic and should be implementable for other protocols representing the same paradigm. Some examples of protocols that are using the publish−subscribe model include XEP−0060,[13] which is an XMPP extension that provides public/subscribe functionalities, and Google Cloud Pub/Sub,[14] which uses HTTPS for the message transport. Other ICCs are more MQTT-specific and to be implementable in other protocols, similar features like retain messages or clear/persistent sessions must exist in them.

Our future work in this area will be devoted to developing countermeasures that will be capable of efficiently detecting and removing the MQTT-based covert channels introduced in this paper.

REFERENCES

[1] W. Mazurczyk, S. Wendzel, Z. Zander, A. Houmansadr, and K. Szczypiorski, *Information Hiding in Communication Networks: Fundamentals, Mechanisms, Applications, and Countermeasures*. Hoboken, NJ, USA: Wiley, 2016.
[2] S. Zander, G. Armitage, and P. Branch, "A survey of covert channels and countermeasures in computer network Protocols," *IEEE Commun. Surveys Tuts.*, vol. 9, no. 3, pp. 44–57, 3rd Quart., 2007.
[3] M. Hron. (2018). *Are Smart Homes Vulnerable to Hacking?* [Online]. Available: https://blog.avast.com/mqtt-vulnerabilities-hacking-smart-homes
[4] K. Cabaj, L. Caviglione, W. Mazurczyk, S. Wendzel, A. Woodward, and S. Zander, "The new threats of information hiding: The road ahead," *IT Prof.*, vol. 20, no. 3, pp. 31–39, May 2018.
[5] W. Mazurczyk and L. Caviglione, "Information hiding as a challenge for malware detection," *IEEE Security Privacy*, vol. 13, no. 2, pp. 89–93, Mar./Apr. 2015.
[6] S. Wendzel, S. Zander, B. Fechner, and C. Herdin, "Pattern-based survey and categorization of network covert channel techniques," *ACM Comput. Surv.*, vol. 47, Apr. 2015, Art. no. 50.
[7] J. S. Kumar and D. R. Patel, "A survey on Internet of Things: Security and privacy issues," *Int. J. Comput. Appl.*, vol. 90, no. 11, pp. 20–26, 2014.
[8] N. Komninos, E. Philippou, and A. Pitsillides, "Survey in smart grid and smart home security: Issues, challenges and countermeasures," *IEEE Commun. Surveys Tuts.*, vol. 16, no. 4, pp. 1933–1954, 4th Quart., 2014.
[9] W. Granzer, W. Kastner, G. Neugschwandtner, and F. Praus, "Security in networked building automation systems," in *Proc. IEEE Int. Workshop Factory Commun. Syst.*, Jun. 2006, pp. 283–292.
[10] C. Müller, F. Armknecht, Z. Benenson, and P. Morgner, "On the security of the ZigBee light link touchlink commissioning procedure," in *Sicherheit 2016—Sicherheit, Schutz Und Zuverlässigkeit*. Bonn, Germany: Gesellschaft für Informatik, 2016, pp. 229–240.
[11] H. Glanzer, L. Krammer, and W. Kastner, "Increasing security and availability in KNX networks," in *Sicherheit 2016—Sicherheit, Schutz Und Zuverlässigkeit*. Bonn, Germany: Gesellschaft für Informatik, 2016.
[12] R. Patuck and J. Hernandez-Castro, "Steganography using the extensible messaging and presence protocol (XMPP)," Sep. 2013, *arXiv:1310.0524*. [Online]. Available: https://arxiv.org/abs/1310.0524
[13] S. Wendzel, "Covert and side channels in buildings and the prototype of a building-aware active warden," in *Proc. IEEE ICC*, Jun. 2012, pp. 6753–6758.
[14] A. Velinov, and D. Stojanov, "New covert channels in Internet of Things," in *Proc. 12th Int. Conf. Emerg. Secur. Inf., Syst. Technol. (SECURWARE)*, Venice, Italy, Sep. 2018, pp. 30–36.
[15] S. Wendzel, W. Mazurczyk, and G. Haas, "Don't you touch my nuts: Information hiding in cyber physical systems," in *Proc. IEEE SPW*, May 2017, pp. 29–34.
[16] C. Rowland. (1997). *Covert Channels in the TCP/IP Protocol Suite*. [Online]. Available: https://firstmonday.org/ojs/index.php/fm/article/view/528/449
[17] I. Zelenchuk. (2004). *Skeeve–ICMP Bounce Tunnel*. [Online]. Available: http://www.gray-world.net/poc_skeeve.shtml
[18] G. Danezis, "Covert communications despite traffic data retention," in *Proc. 16th Int. Workshop Secur. Protocols*, in Lecture Notes in Computer Science, vol. 6615. Cambridge, U.K.: Springer, Apr. 2008, pp. 198–214.
[19] Anonymous. (2005). *DNS Covert Channels and Bouncing Techniques*. [Online]. Available: https://seclists.org/fulldisclosure/2005/Jul/att-452/p63_dns_worm_covert_channel.txt
[20] T. Schmidbauer, S. Wendzel, A. Mileva, and W. Mazurczyk, "Introducing dead drops to network steganography using ARP-caches and SNMP-walks," in *Proc. 3rd Int. Workshop Criminal Use Inf. Hiding (CUING) ARES*, Canterbury, U.K., Aug. 2019, Art. no. 64.
[21] M. Ambrosin, M. Conti, P. Gasti, and G. Tsudik, "Covert ephemeral communication in named data networking," in *Proc. 9th ACM Symp. Inf., Comput. Commun. Secur. (ASIA CCS)*, Kyoto, Japan, Jun. 2014, pp. 15–26.
[22] W. Mazurczyk, S. Wendzel, and K. Cabaj, "Towards deriving insights into data hiding methods using pattern-based approach," in *Proc. 2nd Int. Workshop Criminal Inf. Hiding (CUING) ARES*, Hamburg, Germany, Aug. 2018, Art. no. 10.
[23] A. Banks and R. Gupta. (2015). *MQTT Version 3.1.1 Plus Errata 01*. [Online]. Available: http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/mqtt-v3.1.1.html
[24] A. Banks, E. Briggs, K. Borgendale, and R. Gupta. (2019). *MQTT Version 5.0. OASIS Standard*. [Online]. Available: https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html
[25] S. Wendzel, W. Mazurczyk, and S. Zander, "Unified description for network information hiding methods," *J. Universal Comput. Sci.*, vol. 22, no. 11, pp. 1456–1486, 2016.
[26] L. D. Mills, *Network Time Protocol (NTP)*, document RFC 958, 1985. [Online]. Available: https://tools.ietf.org/html/rfc958
[27] L. D. Mills, *Simple Network Time Protocol (SNTP)*, document RFC 1769, 1995. [Online]. Available: https://tools.ietf.org/html/rfc1769
[28] S. K. Mani, R. Durairajan, P. Barford, and J. Sommers, "MNTP: Enhancing time synchronization for mobile devices," in *Proc. Internet Meas. Conf.*, Santa Monica, CA, USA, Nov. 2016, pp. 335–348.
[29] S. K. Mani, R. Durairajan, P. Barford, and J. Sommers, "An architecture for IoT clock synchronizations," in *Proc. 8th Int. Conf. Internet Things*, Santa Barbara, CA, USA, Oct. 2018, Art. no. 17.
[30] S. Cabuk, C. E. Brodley, and C. Shields, "IP covert channel detection," *ACM Trans. Inf. Syst. Secur.*, vol. 12, no. 4, Pp. 22:1–22:29, 2009.
[31] S. Wendzel, F. Link, D. Eller, and W. Mazurczyk, "Detection of size modulation covert channels using countermeasure variation," *J. Universal Comput. Sci.*, to be published.

[13] https://xmpp.org/extensions/xep-0060.html
[14] https://cloud.google.com/pubsub/docs/

ALEKSANDAR VELINOV received the M.Sc. degree in computer science from the Faculty of Computer Science, University Goce Delcev, Štip, Macedonia, in 2016, where he is currently pursuing the Ph.D. degree. He is also a Teaching Assistant with the Faculty of Computer Science, University Goce Delcev. His research interests include the Internet of Things (IoT), machine-to-machine (M2M), digital steganography, computer and network security, big data, big data analysis, learning analytics, cloud computing, and mobile technologies. He is involved in the following projects: Blended Learning and Developing and Testing and Installing E-learning System for African Member States.

ALEKSANDRA MILEVA received the B.Sc., M.Sc., and Ph.D. degrees from Ss. Cyril and Methodius University in Skopje, Macedonia. She is currently an Associate Professor with the Faculty of Computer Science, University Goce Delcev, Štip, Macedonia, where she is also the Head of the Laboratory of Computer Security and Computer Forensics. Her research interests include computer and network security, digital steganography, the IoT protocols and security, cryptography, computer forensics, and quasi-groups theory. Since 2019, she has been a member of the EURASIP Data Forensics and Security TAC. She was with the management committee of two COST actions IC1201: BETTY and IC1306: Cryptography for Secure Digital Interaction.

STEFFEN WENDZEL received the Ph.D. degree in computer science from the University of Hagen, Germany, in 2013. From 2013 to 2016, he led a Smart Building Security Research Team, Fraunhofer FKIE, Germany. He joined the Worms University of Applied Sciences as a Professor of information security and computer networks, in 2016. Since 2017, he has been the Deputy Scientific Head of the university's Centre for Technology and Transfer. He wrote six books. He has published more than 140 works in journals, such as FGCS, CSUR, *Communications of the ACM*, *Scientometrics*, *Annals of Telecommunications*, and JUCS. His research interests include information hiding and the Internet-of-Things security. He is a member of the Editorial Board of the *Journal of Universal Computer Science* (JUCS) and the *Journal of Cyber Security and Mobility* (JCSM) as well as a Steering Committee Member of CUING. He (co-)organized several workshops and conferences. He served as a Guest Editor for the IEEE TII, IEEE S&P, Elsevier FGCS, Wiley SCN, JUCS, and other journals.

WOJCIECH MAZURCZYK received the B.Sc., M.Sc., Ph.D. (Hons.), and D.Sc. (Habilitation) degrees in telecommunications from the Warsaw University of Technology (WUT), Warsaw, Poland, in 2003, 2004, 2009, and 2014, respectively. He is currently an Associate Professor with the Institute of Computer Science, WUT. He is also a Researcher with the Parallelism and VLSI Group, Faculty of Mathematics and Computer Science, FernUniversitaet, Germany. His research interests include bioinspired cybersecurity and networking, information hiding, and network security. He is involved in the Technical Program Committee of many international conferences and also serves as a Reviewer for major international magazines and journals. Since 2016, he has been the Editor-in-Chief of an open access *Journal of Cyber Security and Mobility*. Since 2018, he has been serving as an Associate Editor for the IEEE TRANSACTIONS ON INFORMATION FORENSICS AND SECURITY and as a Mobile Communications and Networks Series Editor for the *IEEE Communications Magazine*.