Characterization and Evaluation of IoT Protocols for Data Exfiltration

Daniel Uroz, Ricardo J. Rodríguez*, *Member, IEEE*

***Abstract*—Data exfiltration relies primarily on network protocols for unauthorized data transfers from information systems. In addition to well-established Internet protocols (such as DNS, ICMP, or NTP, among others), adversaries can use newer protocols such as Internet of Things (IoT) protocols to inadvertently exfiltrate data. These IoT protocols are specifically designed to meet the limitations of IoT devices and networks, where minimal bandwidth usage and low power consumption are desirable. In this paper, we review the suitability of IoT protocols for exfiltrating data. In particular, we focus on the Constrained Application Protocol (CoAP; version 1.0), the Message Queuing Telemetry Transport protocol (MQTT; in its versions 3.1.1 and 5.0), and Advanced Message Queuing Protocol (AMQP; version 1.0). For each protocol, we review its specification and calculate the overhead and available space to exfiltrate data in each protocol message. In addition, we empirically measure the elapsed time to exfiltrate different amounts of data. In this regard, we develop a software tool (dubbed CHITON) to encapsulate and exfiltrate data within the IoT protocol messages. Our results show that both MQTT and AMQP outperform CoAP. Additionally, MQTT and AMQP protocols are best suited for exfiltrating data, as both are commonly used to connect to IoT cloud providers through IoT gateways and are therefore more likely to be allowed in business networks. Finally, we also provide suggestions and recommendations to detect data exfiltration in IoT protocols.**

***Index Terms*—AMQP 1.0, CoAP 1.0, Data Exfiltration, IoT Protocols, MQTT 3.1.1, MQTT 5.0**

I. INTRODUCTION

INFORMATION is described as one of the most valuable resources in the digital age [1]. Cybercriminals are also aware of this fact and are therefore interested in collecting information, either to obtain valuable assets to trade in the dark market or to obtain advantages against their targets (as is the case with companies with bad practices or state-sponsored hacking groups). As McAffee points out [2], external actors are responsible for the increasing percentage of data theft observed due to breaches, which increased from 57% in 2015 to 61% in 2018.

Computers and networks within organizations are a common target for various types of attacks, ranging from denial of service to phishing [3], [4]. As attacks become more sophisticated, defenses must also keep up. As such, firewalls, demilitarized zone networks, or intrusion detection systems are techniques that are widely deployed in corporate networks as a primary layer of defense against attacks [5].

Unfortunately, cybercriminals are free to collect as much information as possible and exfiltrate all collected data to their servers when these countermeasures fail. The concept of exfiltration comes from the military domain. In the context of computer security, *data exfiltration* refers to the unauthorized transfer of information from an information system [6]. Data exfiltration *per se* does not necessarily indicate any advanced techniques, as a simple outgoing HTTP connection from a compromised email server would essentially accomplish this task. However, some solutions have been proposed to prevent this type of misbehavior, such as Data Loss Prevention (DLP) systems that aim to detect when data is leaving the organization's network [7]. For example, a DLP system can be configured to look up credit card numbers on a company private network, blocking network traffic appropriately if detected.

Consequently, adversaries use techniques such as *covert channels* to avoid detection. A covert channel, first described by [8], is any communication channel that can be exploited by a process to transfer information in a way that violates system security policy [9]. Historically, there is a distinction of the type of covert channels according to how they are used to exfiltrate information [10]: (1) *storage covert channels*, which use values written/read in transmitted objects (e.g., unused header fields); and (2) *timing covert channels*, which use the time of transmitted objects to modulate the information in some way (e.g., a delayed packet means 1 bit, while an on-time packet means 0 bit). In this paper, we focus on storage covert channels.

Similarly, adversaries can use well-established (storage) covert channels in the Internet protocol suite to bypass protections, as they are generally allowed on corporate networks. To name a few examples, adversaries can exfiltrate information through the application-layer Domain Name System (DNS) protocol embedding data in the domain name being queried [11], through the network-layer ICMP protocol embedding data in the binary data transmitted by the echo and echo reply messages [12], or through the application-layer NTP protocol embedding data in the timestamp fields.

Internet of Things (IoT) networks typically make use of a variety of protocols specifically designed to meet the requirements of resource-constrained devices on which they depend, in addition to the common protocols of the Internet protocol

Daniel Uroz and Ricardo J. Rodríguez are with the Department of Computer Science and Systems Engineering, University of Zaragoza, Spain.

*Corresponding author. The authors wish to thank the anonymous referees for providing constructive feedback and helping to improve the content of this manuscript. This work was supported in part by the Aragonese Government under *Programa de Proyectos Estratégicos de Grupos de Investigación* (DisCo research group, ref. T21-20R), and by the University of Zaragoza and the *Fundación Ibercaja* under grant JIUZ-2020-TIC-08. The research of D. Uroz was also supported by the Government of Aragón under a DGA predoctoral grant (period 2019-2023).

suite. IoT is described as the integration of various sensors, objects, and smart nodes capable of communicating with each other without any human intervention [13]. IoT devices can generally be divided into actuators and sensors [14]. An IoT sensor collects information which is then transmitted to a control center to make decisions based on this information and then trigger some kind of physical intervention, which is carried out by an IoT actuator.

IoT devices are generally deployed on isolated and even geographically distant networks thanks to low-power wide area networks, which allow access to devices from thousands of kilometers away [15]. In addition to IoT devices, an IoT network needs an IoT gateway to communicate over the Internet and, in some configurations, with an IoT cloud provider [16]. An IoT gateway is responsible for collecting sensor data, translating between different IoT protocols, and processing data before sending it to an external network, such as the IoT cloud provider's network. In some network configurations, an IoT gateway can also support cellular network technology for communication (for instance, 5G connection), and run on batteries in the event of a power failure. An example of an IoT protocol usually supported by IoT gateways is MQTT 3.1.1, which allows us to connect to AWS IoT Core, Google Cloud IoT Core, and Microsoft Azure IoT Hub cloud providers. The latter cloud provider also supports AMQP 1.0 as IoT protocol.

While traditional covert channels (that is, which rely on protocols from the Internet protocol stack) for data exfiltration are well founded [17], [18], there is very little literature that discusses covert channels in IoT networks [19]–[21]. Recently, the study in [22] analyzes covert channels in the new MQTT 5.0 protocol. Also, to the best of our knowledge, there is no comprehensive comparison of different IoT network protocols for data exfiltration.

In this paper, we identify and review the adequacy of IoT protocols for exfiltrating data. Specifically, we study Constrained Application Protocol (CoAP) v1.0, Message Queuing Telemetry Transport (MQTT) v3.1.1 and v5.0, and Advanced Message Queuing Protocol (AMQP) v1.0. CoAP is a web transfer protocol for use with restricted nodes and networks, specially designed for machine-to-machine (M2M) applications such as smart energy and building automation [23]. MQTT is a a lightweight, open, simple, client-server publish/subscribe messaging transport protocol designed to be easy to implement. These features make it ideal for use in many situations, including constrained environments such as communication in M2M and IoT contexts where a small code footprint is required and/or network bandwidth is a rare commodity [24]. The Advanced Message Queuing Protocol (AMQP) is an open Internet protocol for business messaging that defines a binary wire-level protocol for the reliable exchange of messages between two parties [25].

To fill the gap in the literature, we compare these protocols based on the overhead and the payload available for exfiltrating data of each message. In addition, we compare the time to exfiltrate different data sizes to highlight those types of message that are theoretically valid, but unfeasible in practice. As a side product of our research, we also extend the open source Scapy project [26] to support the AMQP protocol. Finally, to empirically evaluate the performance of data exfiltration using IoT protocols we developed CHITON, a software tool made in Python that allows data exfiltration by encapsulating information in IoT protocol messages.

Contribution. In summary, the contribution of this paper is twofold. First, we provide a comprehensive theoretical study of IoT protocols, from the point of view of data exfiltration. Second, we compare their performance when exfiltrating data in an experimental scenario. In addition, we have developed a software tool that allows data exfiltration by encapsulating information in IoT protocol messages. This tool can be used to evaluate the security mechanisms placed in organizational networks to detect this type of data exfiltration.

Paper organization. Section II presents works related to covert channels techniques and data exfiltration. Section III shows a complete description of the characteristics of the IoT protocols mentioned above and a comparison for their use in data exfiltration. Section IV describes our tool CHITON and the adversary model that we are considering for experimentation. Section V describes various countermeasures that can be applied to defend organization networks against data exfiltration techniques using IoT protocols. Finally, Section VI concludes the paper and suggests directions for future work.

II. Related Work

This section presents related work closely linked to our contributions. We have divided the discussion into three parts: covert channels, data exfiltration, and software tools for data exfiltration.

A. Covert Channels

Covert channels have been widely studied since they were firstly proposed by Lampson [8] as a way of transmitting information between processes that are not allowed to communicate. With the rise of interconnected computers, covert channels quickly adapted to run over network protocols [10].

Rowland was the first to propose covert channels in optional fields of the TCP/IP protocol suite, such as IP packet identification, TCP initial sequence number, or TCP acknowledged sequence number [17]. With the update of the Internet Protocol (IP) to version 6, the use of other fields such as IPv6 extension header has also emerged [27]. A more detailed study can be found in [28], where 22 (storage) covert channels in IPv6 are analyzed.

Covert channels in a variety of protocols like HTTP(S), WLAN, VoIP, SSH, FTP, NTP, to name a few, have also been explored in multiple works. The works in [18], [29], and [30] survey network protocols, techniques, and countermeasures for covert channels. Advanced covert channel have also been proposed, such as covert channels in the physical layer of IEEE 802.3 10 Gigabit Ethernet [31] or covert channels between virtualized systems in the cloud [32].

The tunneling protocol can be seen as a specific type of storage covert channel, where one protocol is embedded within the payload of another protocol. The use of covert channels as tunneling protocols was first proposed using the Internet Control Message Protocol (ICMP) [33] and later using the

Domain Name System (DNS) protocol [34]. In this regard, the `iodine` tool [35] is one of the most mature, studied, and widely-used tools for tunneling through DNS. Although tunneling relies on storage covert channels, it tries to maximize throughput (performance) rather than maintaining low-profile communication (that is, staying undetected for as long as possible). In a more general perspective, other works study malware communication through covert channels [36], [37].

Regarding covert channels in IoT protocols, a recent discussion on CoAP is presented by [21]. Covert channels in wireless sensor networks, modulating transmit power and sensor data, are explored in [38]. In [19], the use of different network destination routes is proposed as timing covert channels. A classification of attack patterns for IoT environments is presented in [39]. It is worth mentioning [20], a survey combining steganography and covert channels in IoT. This work is recently updated in [22], where the use of covert channels in MQTT 5.0 is studied.

Unlike these works, we provide a comprehensive theoretical study of IoT protocols (specifically, CoAP, MQTT, and AMQP) from the point of view of data exfiltration. Let us remark that this approach is not the same as the covert channel approach because data exfiltration does not attempt to breach system security, rather data exfiltration relies on already allowed communications to transfer sensitive information.

B. Data Exfiltration

Most data exfiltration studies focus on air-gapped covert channels, that is, covert channels specially tailored for bridging the gap of physically isolated systems. In this regard, acoustic-based data exfiltration that relies on hard disk noise is presented in [40]. Malware proofs of concept performing data exfiltration through visual channels have also recently been proposed [41], [42]. The use of different radio frequencies as a data exfiltration method has also been studied in [43]. More focused on IoT environments, the work in [44] presents an abuse of smart lighting systems to exfiltrate data through the flickering of light bulbs. Finally, the abuse of an iTunes library to exfiltrate data from paired iOS devices is presented in [45].

Regarding data exfiltration detection, the work in [46] has recently proposed a new model to detect low-throughput data exfiltration through DNS. Similarly, a deep learning-based detection approach for keylogging and data exfiltration attacks is introduced in [47]. Different machine learning models to evaluate their detection rates for different IoT network traffic attack patterns are also used in [48], where authors conclude that data exfiltration detection has the worst metric of all types of attacks (in multiclass classification).

In addition to air-gapped data exfiltration works, we detect a gap in the literature on the analysis and comparison of different protocols to exfiltrate data. To the best of our knowledge, we are the first to thoroughly compare different IoT protocols from a data exfiltration point of view and empirically evaluate their performance when exfiltrating arbitrary data. Hence, our work complements all the aforementioned works.

C. Software Tools for Data Exfiltration

We have searched for tools on GitHub to complete the vision found in the academic literature. Most of the publicly available tools for data exfiltration are intended to evaluate the implementation of Network Monitoring and Data Leakage Prevention configurations. Two tools are the most relevant at the moment of this writing: (i) `DET` [49], which provides support for HTTP, HTTPS, ICMP, DNS, SMTP/IMAP, FTP, and SIP protocols; and (ii) `PyExfil` [50], which supports DNS, ICMP, NTP, BGP, QUIC, Slack, POP3, FTP, IP, and HTTP/S protocols, as well as other physical channels (e.g., audio). In addition, it also incorporates steganographic techniques.

We found other four relevant data exfiltration tools, but all of them are no longer maintained, and were last updated 2 years ago at the time of writing. `IPv6teal` defines a covert channel to exfiltrate data through the IPV6 header field *Flow label* [51]. `IPv6DNSExfil` uses DNS AAAA records to create a command and control channel [52]. Similarly, `DNSExfiltrator` encapsulates data via sub-domain name requests to a domain controlled by an adversary [53]. By default, this tool cuts information into 63-byte chunks (the maximum size of a DNS label) up to a maximum domain of 255 bytes (the maximum length of a DNS domain, including subtags). Finally, the `dnsteal` tools uses a method similar to `DNSExfiltrator` to exfiltrate information [54]. The detection of this DNS exfiltration technique was specifically addressed by [46].

Unlike these tools, CHITON is a modular software library designed to be easily integrated into any other software project, making it easy to use for evaluation and testing. This software library only encapsulates the binary data in protocol messages and, if necessary, sends these messages over the network.

III. Comparative Analysis of IoT Protocols for Data Exfiltration

In this section, we conduct a comparative analysis of IoT protocols for data exfiltration. We first describe the IoT features and protocols used in the comparison and then show their comparative analysis.

A. Comparison Description

To compare them qualitatively, we focus on three characteristics of the protocols: *message type*, which specifies the action to be performed. Each type is made up of payload (how much data a protocol can carry in a single message) and overhead (each byte sent that does not represent exfiltrated data); *transport*, which can be a connectionless transport layer, such as UDP, or a connection-oriented protocol, such as TCP; and *error detection*, which indicates if the protocol counts with any checksum redundancy to detect and correct errors in the received data.

In our approach, the overhead is not just the required message type headers, but we also consider the overhead that each field type introduces. For instance, AMQP string type has a specific constructor to indicate that the following bytes should be interpreted as a string plus its size. This construction means that to exfiltrate 4096 bytes of information,

we introduce a 5-byte overhead, corresponding to one byte to indicate that the following is a Unicode UTF-8 string with length of up to $2^{32} - 1$ bytes and the actual length value of the string, which must be a 4-byte value. Hence, to exfiltrate a 4096-byte payload, we need to send 4101 bytes.

Regarding payload size, it can be refined in theoretical and practical payload size. Due to physical and implementation constraints (*practical*), real world scenarios may require sizes smaller than the sizes defined in a protocol standard (*theoretical*), which may have certain fields with unlimited size. For simplicity of comparison, in this paper we assume ideal scenarios, i.e., we only consider the theoretical payload size.

To compare them quantitatively, we calculate for each protocol the number of messages to be sent, assuming a data transmission of 1MiB. In addition, we consider two types of adversaries: a *stealthy* adversary, who tries to adapt outgoing messages to commonly used sizes; and a *rough* adversary, who maximizes the possible payload for each messages.

Regarding IoT protocols, recall that an IoT gateway allows an IoT device to connect over the Internet (see Section I). Typically, enterprise networks connect the IoT gateway to the network of the cloud service provider, responsible for long-term storage and data analysis. Common protocols offered by IoT cloud service providers are MQTT version 3.1.1 (recently updated to version 5.0), AMQP version 1.0, and CoAP version 1.0. As an adversary can route any data through these protocols directly to the Internet, all of them are considered in this work. Next, we first explain these protocols, and then we highlight the messages that can be used to exfiltrate data, considering all messages types as defined in each protocol specification.

B. IoT Protocol Messages for Data Exfiltration

In what follows, we describe each IoT protocol and show the relevant characteristics of each type of message. We adhere to each protocol specification to name its message types. Message types are typically named depending on the action that is taken. For instance, CoAP calls them *methods*, MQTT *control packets*, and AMQP *performatives*. For each type, we calculate the maximum available payload following a valid message structure. We only rely on message fields whose meaning is user-defined (e.g., tokens) and do not have a well-defined meaning in the specification (e.g., the CoAP *Response Code*, whose value of `4.00` means a `Bad Request`).

Constrained Application Protocol (CoAP)

CoAP is an application layer transfer protocol used with network nodes with restricted capabilities [23]. This protocol follows a request/response scheme (very similar to the HTTP interaction model), where resources are accessed through a URI interface thanks to certain actions on them (called *methods*). Each protocol message consists of mandatory fields such as *Version* or *Message ID*; and optional fields such as *Token*, which are used to match a response to a request; several *Option* fields, similar to HTTP request headers; and a *Payload*. The CoAP reserved port number is 5683.

The `GET` and `DELETE` methods allow data to be sent in the queried URI thanks to the `Uri-Path` *Option*. Although both methods do not contain a CoAP *Payload*, the data can also be extracted via other *Option* fields. In this protocol, the stealthy adversary has several combinations.

Let us explain how the message bytes useful for exfiltrating data are calculated using these methods. First, 1 byte is needed to specify *Version* (2 bits), *Type* (2 bits), and *Token Length* (4 bits). The *Token Length* field establishes the length of the next *Token* field (`1000`, or 8 bytes as the maximum value). The next byte indicates the CoAP method, in the form of `c.dd`, where `c` is represented by the 3 most significant bits and `dd` by the remaining 5 bits. In this case, the *Code* field can be `0.01` (`GET` method) or `0.04` (`DELETE` method). The next 2 bytes indicate the *Message ID* field, which is the first field useful for exfiltrating data, since it does not have a predefined value and is specified by the user. Following a similar reasoning, the next 8 bytes indicate a *Token* to correlate requests and responses, but can also be used to exfiltrate data. After the *Token* field, we can use a total of 5 *Option* fields. Each *Option* is constructed with 1 byte, divided into `Option Delta` (the most significant 4 bits) and `Option Length` (the remaining 4 bits). `Option Delta` indicates the *Option* type and `Option Length` its size (in bytes). Both *Option* fields follow the same structure. When its value is equal to 13, it indicates that the next byte is in fact the real value. When it is 14, it indicates that the actual value is in the next two bytes. The value 15 in `Option Length` is reserved for internal use only, but in `Option Delta` it is reserved for the *Payload Marker*. Using this type of construction, an additional 2 bytes are required for each *Option* used to exfiltrate data. In this case, we have used 5 `Uri-Path` and `Uri-Query` options, 4 of them in the maximum size (255 useful bytes) plus another one of 238 bytes. In total we have an exfiltration payload of 1268 bytes to keep the CoAP message below the IPv6 minimum link MTU of 1280 bytes[1].

Following this structure, we have obtained 1268 bytes of payload for a stealthy adversary with 12 bytes of overhead, resulting in an overhead of 0.94%. Similarly, the rough adversary can use 2 bytes in the *Message ID* field, 8 bytes in the *Token* field, and add different `Uri-Path` and `Uri-Query` options until the payload reaches the maximum of $64,995$ bytes, corresponding to the sum of 254 options of 255 bytes each plus another of 215 bytes.

Other methods such as `POST` and `PUT` allow data to be carried in the body of the message, which also allows data exfiltration. In particular, a body message can contain a CoAP *Payload* after option fields, using a `0xff` byte as an indicator of the start of the payload. Therefore, the stealthy adversary can exfiltrate up to 1273 bytes per message (and thus keep the message size below the IPv6 minimum link MTU) thanks to 2 bytes in the *Message ID* field, 8 bytes in the *Token* field, 255 bytes in the (required) *Uri-Path* option and 1008 bytes in the payload itself, while 2 bytes are used to encode the value of `application/octet-stream` in the `Content-Type` option. In contrast, the rough adversary can exfiltrate $65,500$

[1]The CoAP specification does not enforce an MTU [23]. Instead, when a Path MTU is unknown, the specification recommends assuming an IP MTU of 1280 octets, corresponding to the minimum packet size that must be supported by all nodes in an IPv6 network [55].

bytes in the payload, corresponding to 255 bytes in the Uri-Path option and 65,235 in the payload itself.

Message Queuing Telemetry Transport (MQTT)

MQTT is also an application layer protocol, but implements a publish/subscribe-based messaging transport protocol instead [24]. The latest version of MQTT (version 5.0) introduces several additions, such as adding user properties to the *Variable Header* field. For this protocol, we contemplate that the rough adversary uses these newer properties of MQTT 5.0, while the stealthy adversary is restricted to the older, stable MQTT version 3.1.1 (in which the *Variable Header* field lacks user properties) as version 5 is not currently supported by IoT cloud providers. The MQTT reserved port number is 1883.

CONNECT is the first control packet sent from the client to the server to open a connection. This message contains a variable header, where the Protocol Name field is set to *MQTT* and its payload is a unique ClientID field, encoded in a 23-byte UTF-8 string as maximum length. The stealthy adversary can use this maximum length to exfiltrate data, while the rough adversary can use other user properties (such as Authentication Method, Authentication Data, and various User Property), increasing the useful payload size to 255.99 MiB (thus keeping the MQTT message below the maximum MQTT message size, which is 268,435,455 bytes). The MQTT specification does not enforce a maximum occurrence of User Property fields, regardless of the message type. Therefore, they can be used multiple times until the maximum MQTT message size is reached.

In response to a CONNECT control packet, the server sends the CONNACK control packet. This message contains a variable header, in which Connect Reason Code is set to zero and there is no payload. Hence, CONNACK is not suitable for the stealthy adversary. In contrast, the rough adversary can use up to 255.99 MiB, considering other properties such as Reason String, various User Property, Response Information, Server Reference, Authentication Method, and Authentication Data.

The application messages are sent in PUBLISH control packets, which have a variable header with the Topic Name property encoded as a UTF-8 string of maximum 65,535 bytes, the 2-byte integer field of *Packet Identifier*, and the payload of the message, which can be up to a theoretical value of 268,435,456 bytes (approximately 256 MiB). The stealthy adversary can limit the payload size to the length of an IP packet minus the headers (that is, 65,232 bytes of payload) and limit the use of 255 bytes for the Topic Name field. On the contrary, the rough adversary can use the maximum values of both fields to obtain close to 256 MiB.

The MQTT protocol also supports Quality of Service (QoS) with the PUBACK (QoS 1), PUBREC, PUBREL, and PUBCOMP (QoS 2) control packets. All these messages follow the same structure: a *Packet Identifier* field followed by a *Reason Code* field, which indicates the result of an operation (e.g., value 0x00 for success) and no payload. In this case, the stealthy adversary can exfiltrate only 2 bytes in the *Packet Identifier* field, while the rough adversary can use a payload of up to 255.99 MiB via the Reason String and various User Property properties.

The SUBSCRIBE control packet is sent from the client to the server to indicate a subscription. The variable header of this message has a *Packet Identifier* field, which is a 2-byte integer, and a pair of *Topic Filter* field as payload, which is a UTF-8 encoded string of up to 65,535 bytes and a *Subscription Options* field, which is one byte in size. Likewise, the UNSUBSCRIBE control packet is sent to unsubscribe from topics. This message is also made up of the *Packet Identifier* field and the *Topic Filter* field, which in this case describes the list of topic filters from which the client wishes to unsubscribe. With these messages, the stealthy adversary can use the *Packet Identifier* and the *Topic Filter* fields to exfiltrate up to 257 bytes, while the rough adversary can use the full size of the *Topic Filter* field plus various *User Property* fields, raising the payload to a value of 255.99 MiB.

The server responds to the SUBSCRIBE control packet sent by the client with a SUBACK control packet, confirming receipt and processing of the subscription. Also, the UNSUBACK control packet is sent in response to the UNSUBSCRIBE control packet. Both acknowledge message have a variable header that has the field *Packet Identifier*, while the message payload contains a list of Reason Codes values (one byte each) corresponding to all possible subscription topics in the SUBSCRIBE control packet. A stealthy adversary can exfiltrate 2 bytes per message using the *Packet Identifier* field, whereas the rough adversary can exfiltrate up to 255.99 MiB using the Reason String and User Property properties.

PINGREQ and PINGRESP are ping-like control packets sent from a client to check if the server is up. These messages, though, are not suitable for exfiltrating data as they do not have variable header or payload fields.

The DISCONNECT control packet is the last MQTT control packet sent from the client or the server, and specifies the reason for closing the network connection in the *Disconnect Reason Code* field (1 byte long), without any payload field. As with the CONNACK control packet, the stealthy adversary cannot use it while the rough adversary can exfiltrate up to 255.99 MiB by means of the Reason String, various User Property, and Server Reference properties.

The last message is the AUTH control packet, which is sent as part of an extended authentication exchange (such as challenge/response authentication). Similar to the DISCONNECT control packet, the variable header of this message has a Authenticate Reason Code property (1 byte long), without any payload field. As before, it is not suitable to be used by the stealthy adversary, whereas the rough adversary can exfiltrate up to 255.99 MiB thanks to the Authentication Method, Authentication Data, Reason String, and various User Property properties.

Advanced Message Queuing Protocol (AMQP)

AMQP is an application layer protocol for message-oriented middleware software infrastructures [25]. It is a peer-to-peer protocol at the binary wire level to transport messages between two processes over a network. An AMQP encoded data stream comprises bytes of unknown type with embedded constructors, which indicate how the bytes should be interpreted (i.e., its type is defined within the data). AMQP is a complex protocol that has more than 24 native data *types* (such as `string` or `boolean`), plus 19 specific definitions built on these native data types. Since binary encoding is not ideal for representing types, the type notation in AMQP follows an XML schema. The AMQP reserved port number is 5672. AMQP is a complex protocol and its library implementations have a large footprint, so they are generally used by devices whose resources are not constrained, such as IoT gateways.

AMQP messages comprise a *Frame Header* field of a fixed-length 8-byte structure preceding each frame, a variable-length *Extended Header* field for future extensions of the protocol, and a variable-length *Frame Body* field containing a byte sequence, the format of which depends on the type of frame. Each type of AMQP message corresponds to an AMQP action (called *performative*). In this case, we consider that the stealthy adversary only uses the mandatory fields of each performative, while the rough adversary uses all possible fields.

The `Open` performative negotiates the connection parameters. The stealthy adversary can use the *container-id* field to exfiltrate up to 4096 bytes (we consider the Linux `PATH_MAX` size[2] as the upper limit for string types), while the rough adversary can use also the *hostname*, *outgoing-locales*, *incoming-locales*, *offered-capabilities*, *desired-capabilities*, *properties* fields in addition to *container-id* to exfiltrate up to 4 GiB. Each of these fields can hold up to 4 GiB except the *properties* field, which can contain up to 8 GiB, resulting in a total of 32 GiB. Unfortunately, this number is bounded by the upper bound of the `Open` list descriptor, which is 4 GiB. This limitation also applies to all other AMQP performatives.

The `Begin` performative begins a session on a channel. In this case, the stealthy adversary can use a 12-byte payload, spread over 4 bytes of each of the *next-outgoing-id*, *incoming-window*, and *outgoing-windows* fields. In contrast, the rough adversary can maximize the payload to be up to 4 GiB using *offered-capabilities*, *desired-capabilities*, and *properties* fields.

The `Attach` performative binds a link to a session. The stealthy adversary can use a payload of 4100 bytes, distributed as 4096 bytes in the *name* field plus 4 bytes in the *handle* field. As before, the rough adversary can maximize the length of data exfiltrated up to 4 GiB by using *name*, *unsettled*, *offered-capabilities*, *desired-capabilities*, and *properties* fields plus the use of the `string` type with the *source* and *target* fields, which use wildcard data types ($\star$), expressing that they can contain any type of data.

The `Flow` performative updates the flow state for the specified link. This performative allows the stealthy adversary to exfiltrate 12 bytes, distributed over 4 bytes in each of the *incoming-window*, *next-outgoing-id*, and *outgoing-window* fields. In contrast, the rough adversary can maximize the length of exfiltrated data to 4 GiB thanks to the *properties* field.

The `Transfer` performative sends messages through a link. In this case, the stealthy adversary can use a 4-byte payload via the *handle* field, while the rough adversary can again get 4 GiB for exfiltrating data by relying on the *state* field and using any data as the wildcard type that support that size. Furthermore, both adversaries can also use the message payload for their data exfiltration purposes. The stealthy adversary can add up to $65,471$ bytes to reach the maximum IP packet length ($65,535$ bytes). In contrast, the rough adversary can also use it to transfer up to 4 GiB of data more.

The `Disposition` performative informs the remote peer of local changes in delivery states. In this case, the stealthy adversary can use a 4-byte payload in the *first* field, while the rough adversary, as before, can maximize the payload up to 4 GiB thanks to the *state* field with any other type of data that supports that size.

As for the message sent to end the connections, there are three performatives. The `Detach` performative detaches the link endpoint from a previously established session. With this performative, the stealthy adversary can use only a 4-byte payload in the *handle* field, while the rough adversary can exfiltrate up to 4 GiB just using the *error* field. The `End` performative indicates that the session has ended, while the `Close` performative indicates that the sender will not send any more frames on that connection. Both performatives have no required fields and therefore they are not eligible candidates for the stealthy adversary. In contrast, the rough adversary can still rely on the *error* field, exfiltrating up to 4 GiB data.

C. Discussion

Regarding qualitative analysis, CoAP relies on UDP as the transport layer because it consumes less resources on the devices, unlike MQTT and AMQP, which are based on TCP. None of the protocols considered in this paper performs any type of integrity verification to detect transmission errors, relying exclusively on the error detection mechanisms provided by the transport layers (both UDP and TCP have a checksum field of 16 bytes).

Regarding quantitative analysis, Table I summarizes the results of both adversaries. For each message, we show the total message size (that is, payload plus overhead), the percentage of overhead relative to the total size, and the number of messages required to encapsulate 1 MiB of data. For each protocol, we highlight in the table the most suitable message type for data exfiltration, that is, the one with the fewest messages and the least overhead. As expected, we can see that the best message to exfiltrate data for each protocol is the message designed to transport information between communication nodes. Specifically, `POST` and `PUT` for CoAP, `PUBLISH` for MQTT, and `Transfer` for AMQP.

Comparing between protocols, AMQP and MQTT stand out for rough adversaries, since they only need one message of any type to fully exfiltrate 1 MiB of data

[2]See https://github.com/torvalds/linux/blob/master/include/uapi/linux/limits.h (visited on August 26, 2021).

(except PINGREQ/PINQRESP). Based on the number of messages, CoAP is the most stable protocol since the number of messages required varies less, regardless of the type of message. For a stealthy adversary, MQTT needs on average twice as many messages compared to AMQP because PUBACK/PUBREC/PUBREL/PUBCOMP and SUBACK/UNSUBACK can only exfiltrate 5 bytes per message. Still, the MQTT PUBLISH control packet is the best message of all protocols, requiring fewer messages and with an overhead of only 6 bytes for a stealthy adversary and 7 bytes for a rough adversary.

In terms of overhead, CoAP introduces the minimum overhead among all the packages studied. For a stealthy adversary, MQTT introduces a third less overhead (on average) compared to AMQP. However, when the data to be exfiltrated increases, AMQP's binary encoding provides more size-optimized results for a rough adversary, producing 32 bytes of overhead on average, with only 20 bytes of overhead at best (Transfer performative) and 49 bytes of overhead in the worst case (Flow performative).

IV. Experiments and Discussion

In this section, we first introduce CHITON, a software tool developed as a side-product in this research to encapsulate information in IoT protocol messages. We then model the adversary scenario used later in our experiments and finally discuss the experimental results.

A. The CHITON Software Tool

In the context of this research, we have developed a software artifact, dubbed CHITON, to empirically evaluate data exfiltration on IoT protocols. This tool allows the user to encapsulate arbitrary data in messages of the IoT protocols explained in Section III. In the interest of open science, we have released CHITON as open source under the Affero GPL version 3 license [56].

CHITON is developed in the Python programming language to support multiple platforms. In particular, our tool relies on Scapy [26], which is a well-established and mature packet manipulation Python library that allows the user to manipulate network packets and automatically convert them to/from packet objects to their (on-wire) byte representation very simply. Scapy provides support for some IoT protocols, such as CoAP and MQTT. However, the AMQP protocol is not supported. In addition, and as required by CHITON, we also added a new functionality in Scapy to support the AMQP protocol.

In addition to being a (standalone) tool, CHITON is also provided as a software library, making it easy to integrate into other tools and analysis workflows. Our tool follows a client/server architecture: the network host that wants to extract data acts as a client, while the network host that receives it acts as a server. Before data exfiltration occurs, both hosts have to agree (in some way) on which protocol and which protocol message will be used.

B. Adversary Model

As far as we know, the use of IoT protocols for data exfiltration is not a common practice in cybercrime today. However, the prevalence of IoT devices in organizational networks increases the attack surface and can cause this to change in a short time. For example, the recent T-Mobile breach occurred when an attacker gained access to a GPRS gateway that was allegedly misconfigured and exposed to the Internet, allowing the attacker to eventually switch to the LAN [57]. A similar issue may occur in organizational networks in which IoT devices are placed.

As an example of a network, we consider a typical business network segregated into two subnets, a *corporate IoT network* made up of heterogeneous devices, such as IoT devices and workstations connected to an IoT gateway, and the *IoT cloud network*, responsible for long-term storage and data analysis. Regarding the adversary model, we consider an adversary with basic capabilities (*basic adversary*), such as a script kiddie [58] who aims to maximize the payload exfiltrated in each message and use IoT protocols trying to stay undetected. The behavior of this adversary corresponds to the *rough adversary*, previously defined in Section III-A. In this adverse scenario, there is at least one compromised workstation within the corporate network. For example, a script kiddie would be an adversary who gains access to a workstation by spear phishing credentials.

An adversary will choose IoT traffic over traditional TCP/IP traffic (such as SMTP, DNS, or HTTP, among others) in an attempt to avoid heavily monitored networks. Furthermore, IoT protocols may be the only alternative for external communication in IoT networks. The use of IoT protocols also has some limitations, as the adversary limits their attack to those organizations that have IoT networks in place. In addition, constrained IoT devices can overheat if data exfiltration is prolonged in time, which can cause hardware failures that generate alarms in the attacked organization, thus making it easier to discover the presence of the adversary.

We consider that this adversary does not have privileged access on any of the compromised devices within the corporate IoT network, so only ports greater than 1024 are available to communicate. Note that ports below 1024 are reserved for the system and cannot be used without privileged permissions. This restriction clearly limits the data exfiltration technique to be used because the adversary would have to act as a client, restricting the protocol messages available for tunneling. For example, while a DNS request (client) has a payload of up to 245 bytes, a DNS response (server) can contain up to the theoretical limit of $65,156$ bytes. This is the reason for the (asymmetric) limitation of the upstream bandwidth when extracting data through the traditional [35] protocols.

C. Experiments

Our goal is to empirically measure how long the data exfiltration takes using the IoT protocols presented in Section III. The data size ranges from 1 KiB to $100,000$ KiB to assess how well each IoT protocol performs when the data size is increased.

Table I
EXFILTRATION OF 1,048,576 BYTES (1 MIB) BY IOT PROTOCOL. MESSAGE SIZE IS EXPRESSED IN BYTES.

	Stealthy Adversary			Rough Adversary		
	Message Size	**Overhead**	**# Messages**	**Message Size**	**Overhead**	**# Messages**
CoAP						
GET/DELETE	1280	0.94%	827	65,507	0.74%	17
POST/PUT	1280	0.55%	820	65,507	0.01%	17
MQTT		(***version 3.1.1***)			(***version 5.0***)	
CONNECT	37	37.84%	45,591	1,048,635	0.01%	1
CONNACK	-	-	-	1,048,628	$< 0.01\%$	1
PUBLISH	65,495	0.01%	17	1,048,583	$< 0.01\%$	1
PUBACK/PUBREC/PUBREL/PUBCOMP	5	60%	524,288	1,048,627	$< 0.01\%$	1
SUBSCRIBE	263	2.28%	4081	1,048,626	$< 0.01\%$	1
UNSUBSCRIBE	262	1.91%	4081	1,048,625	$< 0.01\%$	1
SUBACK/UNSUBACK	5	60%	524,288	1,048,627	$< 0.01\%$	1
PINGREQ/PINGRESP	-	-	-	-	-	-
DISCONNECT	-	-	-	1,048,627	$< 0.01\%$	1
AUTH	-	-	-	1,048,627	$< 0.01\%$	1
AMQP						
Open	4121	0.61%	256	1,048,601	$< 0.01\%$	1
Begin	30	60%	87,382	1,048,606	$< 0.01\%$	1
Attach	4126	0.63%	256	1,048,602	$< 0.01\%$	1
Flow	30	60%	87,382	1,048,625	$< 0.01\%$	1
Transfer	65,495	0.03%	17	1,048,596	$< 0.01\%$	1
Disposition	20	80.00%	262,144	1,048,605	$< 0.01\%$	1
Detach	19	78.95%	262,144	1,048,615	$< 0.01\%$	1
End	-	-	-	1,048,613	$< 0.01\%$	1
Close	-	-	-	1,048,613	$< 0.01\%$	1

Since the nature of the data file is irrelevant, we generate a random file of the maximum payload length (100 MiB) and divide it into chunks of the desired size. To verify the integrity of transmitted data, we compute the SHA256 hash of each chunk on the client and server sides and compare them. In all of our experiments, the matching of the calculated hashes was successful.

Transmission time is measured on the server side, as the server is responsible for receiving the exfiltrated data. Each message is identified during transmission thanks to a message counter: the first message has the value 1 and the last one has the value 0. Therefore, the start time is taken when the first message arrives, while the end time is taken when the last message arrives. Consequently, the elapsed time is calculated subtracting these times. Note that network latency plays a very important role in this type of measurement. To overcome this, we repeat the experiments ten times at different times during a day and then calculate the average.

We simulate the aforementioned basic adversary model by interconnecting two network devices through the Internet: one host (the client) is located in the network of the University of Zaragoza, while the other host (the server) is located in the home office of the main author of this paper. The distance between them is 14 hops, measured by the `traceroute` tool, and the average latency is 21.546 ms, measured by the `ping` utility. The client device runs a Raspbian Buster on top of a Raspberry Pi 3 Model B, which plays the role of a compromised constrained workstation. This device has a 1.2 GHz Broadcom BCM2837 processor, 1 GB of LPDDR2 RAM (900 MHz), and a 100Base-T Ethernet connection. The server device is a personal computer (PC), which plays the role of malicious server that receives the exfiltrated data. This computer runs Ubuntu 20.04 on an 4.60 GHz Intel i7-8700 processor, 16GB of DDR4 RAM (2400 MHz), and a 1000Base-TX Ethernet connection with a RTL8111 chipset.

In this scenario, the Raspberry runs a client instance of CHITON and is responsible for exfiltrating all data, while the PC runs the server instance of CHITON to receive the exfiltrated data over the network. We assume that the adversary previously obtained access to the workstation and was able to install the CHITON client. The PC is also capturing the network traces with `tcpdump` for debugging purposes. Regarding the protocol messages, we only consider the best messages to exfiltrate data for each protocol, as shown in Table I. We consider a message best suited for exfiltrating data when it shows the highest payload/overhead ratio. In particular, we select the `POST` and `PUT` methods (from CoAP), the `PUBLISH` control packet (for MQTT), and the `Transfer` performative (for AMQP). A detailed explanation of these messages is given in Section III.

D. Discussion of Results

The experimental results using the best message size of `CoAP`, `MQTT`, and `AMQP` protocols are plotted in Figure 1. Note that both the file size axis (in KiB) and the transmission time axis (in milliseconds) are on logarithmic scale. As shown, the transmission times of the MQTT and AMQP protocols are identical in all cases. On the contrary, there is a notable difference with regard to the transmission time of the CoAP protocol. This difference can be motivated by three reasons:

1) *In CoAP protocol, more messages are needed to send the same amount of data*. This implies a longer time overhead because the operating system, together with the network controller, must deal with the underlying I/O network operations more frequently.
2) *The CoAP protocol runs over UDP, unlike the MQTT and AMQP protocols*. Note that depending on the configuration of the intermediate network nodes, packet traffic

policies can be implemented to prioritize TCP over UDP traffic. Therefore, intermediate nodes could deliberately delay UDP traffic. Unfortunately, this problem cannot be mitigated by the basic adversary.

3) *CoAP message size could be suboptimal*. As stated in Section III, despite using the 1280-byte IP upper limit recommended by the standard CoAP specification [23], this message size might not be the most appropriate value for the network. While TCP implements a retransmission mechanism for lost packets, an application that relies on UDP protocol needs to detect packet loss at the end node and notify the source node to retransmit lost packets appropriately. This packet loss could be due to many reasons, such as network congestion, device computing performance, or wireless network interference, to name a few. In the same way, nodes communicating in a network could support larger packets before incurring packet loss, so the optimal size of the CoAP message could be greater than the limit applied in our experiments and therefore increase the efficiency of communication.

In general, TCP incurs network overhead to provide key features such as orderly data transfer or retransmission of lost packets. In particular, TCP causes an overhead in the establishment and termination of a connection due to the three-way handshake and four-way handshake, respectively. This connection-oriented overhead is negligible because this number of packets is much less than the actual number of messages transmitted to exfiltrate the data.

Finally, let us note that we found no problem with the loss or disorder of UDP data during our experiments. However, UDP is a best-effort transport layer, which means that packets in transit are likely to get dropped or out of order for a myriad of reasons. Therefore, the client would need to resend all the data in the event of a single packet loss due to the client implementation of CHITON, drastically increasing the exfiltration time. At the moment of this writing, CHITON detects UDP packet loss or disorder, but does not have any packet replay mechanism. As future work, our goal is to incorporate a replay packet mechanism in our tool to overcome these issues.

V. COUNTERMEASURES

In this section, we provide suggestions and recommendations for network engineers to detect the use of IoT protocols as covert channels for data exfiltration.

Zero Trust Architecture (ZTA). Traditional networks have evolved to inverted networks where the perimeter itself has dissolved due to the proliferation of cloud computing, mobile device use, and the IoT. This evolution has caused a paradigm shift as traditionally, nation-state agencies and enterprise networks are (in general) only focused on their perimeter defense and coarse-grained subject permissions. As a result, one of the biggest challenges in these networks (and for nation-state agencies in particular) has been unauthorized lateral movements within the environment.

Zero trust architecture is an enterprise-architectural approach to prevent data breaches and limit internal lateral movement [59]. Based on zero trust principles (roughly speaking, principles of "never trust, always verify"), it is designed to reduce the risk exposure of the enterprise in the new perimeterless world. ZTA entails securing the access to resources from any location and controlling it on a need-to-know basis (strictly enforced). In addition, it forces the organizations to inspect and log all traffic to audit whether their users are doing the right thing.

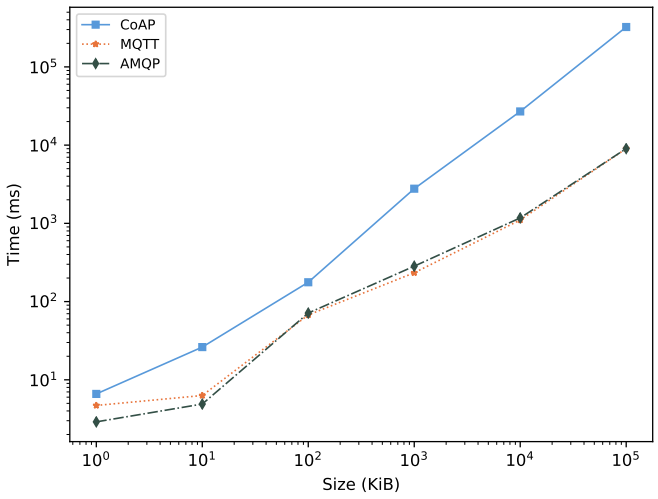

Figure 1. Exfiltration times of data of different file sizes (using the best message type of each protocols).

Stateful-based detection. Stateful protocol analysis involves comparing predetermined profiles of benign (generally accepted) protocol states with observed events to identify deviations. Some vendors use the term *deep packet inspection* to denote some type of stateful protocol analysis, often combined with a firewall ability to block any detected malicious communication. Henceforth, we use the term *stateful protocol analysis* as it is more appropriate to refer to parsing of both network-based and host-based activities, whereas deep packet inspection refers only to the network-based activity analysis.

Unlike anomaly-based detection, which uses host or network-specific profiles, stateful protocol analysis relies on vendor-developed universal profiles that specify how a protocol will behave. Therefore, it can identify unexpected sequences of protocol commands, such as issuing a command multiple times or issuing a command out of the correct order. These profiles are built upon protocol models, which are based primarily on protocol definitions from software vendors and standards entities.

Stateful detection systems are (advanced) intrusion detection and prevention systems capable of understanding and tracking the network, transport, and application protocols to get a sense of a *global state* [60]. These systems also include reasonableness checks for individual commands, such as minimum and maximum lengths for command arguments. For instance, when a command typically has a username argument whose maximum length is 20 characters, an argument of 1000 characters is suspect. Machine-learning approaches can also be used to detect undesired communications.

This solution has some limitations, though. With regard

to protocol models, many standards do not fully explain the internal details of protocols, causing variations between vendor implementations. Additionally, many vendors violate standards or extend them with proprietary features, some of which may even replace standard features. Also, full details on proprietary protocols are often unavailable, making a complete and accurate analysis difficult. Since standard protocols are continually reviewed and vendors modify their protocol implementations to accommodate these changes, protocol models must also be updated to reflect them.

Regarding protocol analysis, the analysis task is a resource intensive task due to the complexity of the analysis and the overhead involved in tracking status for many simultaneous sessions. Moreover, attacks that do not violate acceptable protocol behavior, such as taking many benign actions in a very short period of time to cause a denial of service, are not detected. In addition to these problems, particular versions of specific applications and operating systems within the network may have different protocol implementations, which can be a major restriction for analysis.

VI. Conclusions and Future Work

In this work, we studied the CoAP, MQTT, and AMQP protocols from the point of view of data exfiltration. Data exfiltration is the unauthorized transfer of information from an information system. Furthermore, we also introduced CHITON, a Python library that allows data to be exfiltrated through the three aforementioned protocols. To the best of our knowledge, this work is the first to broadly compare these IoT protocols from a data exfiltration point of view, focusing on characteristics such as overhead and available payload to exfiltrate data in each protocol message.

In addition, we empirically measured and compared the time required to exfiltrate files of different data sizes. The experimentation shows that CoAP is the least suitable protocol and that both MQTT and AMQP outperform it. This preference for MQTT and AMQP is also supported from an adversary's point of view, as these protocols are more likely to be allowed on the enterprise networks because they are typically used to connect enterprise IoT networks to IoT cloud providers. For instance, MQTT version 3.1.1 is an IoT protocol supported by the three major cloud providers (Amazon Web Services, Microsoft Azure, and Google Cloud).

We believe this work can be taken as a basis to study how to detect an unintended data transfer using IoT protocols and therefore improve detection systems appropriately. In this regard, we discussed the use of network segmentation, stateful intrusion detection systems, and deep packet inspection techniques against the problem of data exfiltration using IoT protocols. As future work, we aim to broaden the scope of this study by adding more protocols (such as MQTT-SN, a version of MQTT specially designed for the IoT environment) and measuring the performance impact on various IoT devices. Regarding CHITON, our goal is to add support for a response mechanism for lost UDP packets and for packet sorting, as well as to extend the number of supported protocols.

References

[1] K. Schwab, A. Marcus, J. Oyola, W. Hoffman, and M. Luzi, "Personal Data: The Emergence of a New Asset Class," [Online; http://www3.weforum.org/docs/WEF_ITTC_PersonalDataNewAsset_Report_2011.pdf], World Economic Forum, Tech. Rep., 2011, accessed on May 11, 2021.

[2] McAfee, "Grand Theft Data II: The Drivers and Shifting State of Data Breaches," [Online; https://www.mcafee.com/enterprise/en-us/assets/executive-summaries/es-data-exfiltration-2.pdf], MSI-ACI Europe, Tech. Rep., 2019, accessed on May 11, 2021.

[3] B. B. Gupta, R. C. Joshi, and M. Misra, "Defending against Distributed Denial of Service Attacks: Issues and Challenges," *Information Security Journal: A Global Perspective*, vol. 18, no. 5, pp. 224–247, 2009.

[4] B. B. Gupta, A. Tewari, A. K. Jain, and D. P. Agrawal, "Fighting against phishing attacks: state of the art and future challenges," *Neural Computing and Applications*, vol. 28, no. 12, pp. 3629–3654, 12 2017.

[5] J. M. Kizza, *Guide to Computer Network Security*, 5th ed. Springer, Cham, 2020.

[6] NIST, "Glossary: Exfiltration," [Online; https://csrc.nist.gov/glossary/term/exfiltration], 2020, accessed on May 20, 2021.

[7] T. Takebayashi, H. Tsuda, T. Hasebe, and R. Masuoka, "Data Loss Prevention Technologies," *FUJITSU Sci. Tech. J.*, vol. 46, no. 1, pp. 47–55, 1 2010.

[8] B. W. Lampson, "A Note on the Confinement Problem," *Commun. ACM*, vol. 16, no. 10, p. 613–615, Oct. 1973.

[9] U.S. Department Of Defense, *Trusted Computer System Evaluation Criteria*. London: Palgrave Macmillan UK, 1985, pp. 1–129.

[10] C. Girling, "Covert Channels in LAN's," *IEEE Transactions on Software Engineering*, vol. 13, no. 02, pp. 292–296, feb 1987.

[11] R. J. Rodríguez, "Evolution and Characterization of Point-of-Sale RAM Scraping Malware," *Journal in Computer Virology and Hacking Techniques*, vol. 13, no. 3, pp. 179–192, Aug. 2017.

[12] L. Macrohon and R. Mendrez, "Pingback: Backdoor At The End Of The ICMP Tunnel," [Online; https://www.trustwave.com/en-us/resources/blogs/spiderlabs-blog/backdoor-at-the-end-of-the-icmp-tunnel/], accessed on November 30, 2021.

[13] M. Conti, A. Dehghantanha, K. Franke, and S. Watson, "Internet of Things security and forensics: Challenges and opportunities," *Future Generation Computer Systems*, vol. 78, no. 2, pp. 544 – 546, 2018.

[14] A. Rayes and S. Salam, *The Things in IoT: Sensors and Actuators*. Cham: Springer International Publishing, 2017, pp. 57–77.

[15] S. Balaji, K. Nathani, and R. Santhakumar, "IoT Technology, Applications and Challenges: A Contemporary Survey," *Wireless Personal Communications*, vol. 108, no. 1, pp. 363–388, 9 2019.

[16] H. Chen, X. Jia, and H. Li, "A brief introduction to IoT gateway," in *IET International Conference on Communication Technology and Application (ICCTA 2011)*, 2011, pp. 610–613.

[17] C. H. Rowland, "Covert channels in the TCP/IP protocol suite," *First Monday*, vol. 2, no. 5, 1997.

[18] S. Zander, G. Armitage, and P. Branch, "A Survey of Covert Channels and Countermeasures in Computer Network Protocols," *IEEE Communications Surveys & Tutorials*, vol. 9, no. 3, pp. 44–57, 2007.

[19] I. S. Moskowitz, S. Russell, and B. Jalaian, "Steganographic Internet of Things: Graph Topology Timing Channels," in *The Workshops of the The Thirty-Second AAAI Conference on Artificial Intelligence, New Orleans, Louisiana, USA, February 2-7, 2018*, ser. AAAI Workshops, vol. WS-18. AAAI Press, 2018, pp. 252–259.

[20] A. Mileva, "Steganography in the World of IoT," [Online; https://eprints.ugd.edu.mk/id/eprint/20424], 2018, accessed on May 20, 2021.

[21] A. Mileva, A. Velinov, and D. Stojanov, "New covert channels in Internet of Things," in *The 12th International Conference on Emerging Security Information, Systems and Technologies*, 2018, pp. 30–36.

[22] A. Mileva, A. Velinov, L. Hartmann, S. Wendzel, and W. Mazurczyk, "Comprehensive analysis of MQTT 5.0 susceptibility to network covert channels," *Computers & Security*, vol. 104, p. 102207, 2021.

[23] Z. Shelby, K. Hartke, and C. Bormann, "The Constrained Application Protocol (CoAP)," Internet Engineering Task Force (IETF), RFC 7252, June 2014, accessed on May 11, 2021. [Online]. Available: https://tools.ietf.org/rfc/rfc7252.txt

[24] A. Banks, E. Briggs, K. Borgendale, and R. Gupta, "MQTT Version 5.0," OASIS, Standard, March 2019, accessed on May 11, 2021. [Online]. Available: https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html

[25] R. Godfrey, D. Ingham, and R. Schloming, "OASIS Advanced Message Queuing Protocol (AMQP) Version 1.0," [Online; http://docs.oasis-open.org/amqp/core/v1.0/os/amqp-core-complete-v1.0-os.pdf], OASIS Standard, Tech. Rep., October 2012, accessed on May 11, 2021.

[26] P. Biondi, "Scapy Project," [Online; https://scapy.net], 2005, accessed on May 20, 2021.

[27] T. Graf, "Messaging over IPv6 destination options," [Online; https://web.archive.org/web/20200208203832/http://gray-world.net/papers/messip6.txt], 2003, accessed on May 20, 2021.

[28] N. B. Lucena, G. Lewandowski, and S. J. Chapin, "Covert Channels in IPv6," in *Privacy Enhancing Technologies*, G. Danezis and D. Martin, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 147–166.

[29] D. Llamas, C. Allison, and A. Miller, "Covert channels in internet protocols: A survey," in *Proceedings of the 6th Annual Postgraduate Symposium about the Convergence of Telecommunications, Networking and Broadcasting, PGNET*, vol. 2005, 2005.

[30] W. Mazurczyk, S. Wendzel, S. Zander, A. Houmansadr, and K. Szczypiorski, *Information Hiding in Communication Networks: Fundamentals, Mechanisms, Applications, and Countermeasures*, 1st ed. Wiley-IEEE Press, 2016.

[31] K. S. Lee, H. Wang, and H. Weatherspoon, "PHY Covert Channels: Can you see the Idles?" in *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)*. Seattle, WA: USENIX Association, Apr. 2014, pp. 173–185.

[32] Z. Wu, Z. Xu, and H. Wang, "Whispers in the Hyper-Space: High-Bandwidth and Reliable Covert Channel Attacks Inside the Cloud," *IEEE/ACM Transactions on Networking*, vol. 23, no. 2, pp. 603–615, 2015.

[33] daemon9, "Project Loki," *Phrack Magazine*, vol. 7, no. 49, August 1996.

[34] O. Pearson, "DNS Tunnel - through bastion hosts," [Online; https://web.archive.org/web/20200208203702/https://www.gray-world.net/papers/dnstunnel.txt], 1998, accessed on May 20, 2021.

[35] E. Ekman, "iodine," [Online; https://code.kryo.se/iodine/], 2006, accessed on May 20, 2021.

[36] W. Mazurczyk and L. Caviglione, "Information Hiding as a Challenge for Malware Detection," *IEEE Security & Privacy*, vol. 13, no. 2, pp. 89–93, 2015.

[37] K. Cabaj, L. Caviglione, W. Mazurczyk, S. Wendzel, A. Woodward, and S. Zander, "The New Threats of Information Hiding: The Road Ahead," *IT Professional*, vol. 20, no. 3, pp. 31–39, 2018.

[38] N. Tuptuk and S. Hailes, "Covert channel attacks in pervasive computing," in *2015 IEEE International Conference on Pervasive Computing and Communications (PerCom)*, 2015, pp. 236–242.

[39] L. Caviglione, A. Merlo, and M. Migliardi, "Covert Channels in IoT Deployments Through Data Hiding Techniques," in *2018 32nd International Conference on Advanced Information Networking and Applications Workshops (WAINA)*, 2018, pp. 559–563.

[40] M. Guri, Y. Solewicz, A. Daidakulov, and Y. Elovici, "Acoustic Data Exfiltration from Speakerless Air-Gapped Computers via Covert Hard-Drive Noise ('DiskFiltration')," in *Computer Security – ESORICS 2017*, S. N. Foley, D. Gollmann, and E. Snekkenes, Eds. Cham: Springer International Publishing, 2017, pp. 98–115.

[41] A. Robles-Durazno, N. Moradpoor, J. McWhinnie, and G. Russell, "WaterLeakage: A Stealthy Malware for Data Exfiltration on Industrial Control Systems Using Visual Channels," in *2019 IEEE 15th International Conference on Control and Automation (ICCA)*, 2019, pp. 724–731.

[42] E. Carpentier, C. Thomasset, and J. Briffaut, "Bridging The Gap: Data Exfiltration In Highly Secured Environments Using Bluetooth IoTs," in *2019 IEEE 37th International Conference on Computer Design (ICCD)*, 2019, pp. 297–300.

[43] M. Guri, G. Kedma, A. Kachlon, and Y. Elovici, "AirHopper: Bridging the air-gap between isolated networks and mobile phones using radio frequencies," in *2014 9th International Conference on Malicious and Unwanted Software: The Americas (MALWARE)*, 2014, pp. 58–67.

[44] E. Ronen and A. Shamir, "Extended Functionality Attacks on IoT Devices: The Case of Smart Lights," in *2016 IEEE European Symposium on Security and Privacy (EuroSP)*, 2016, pp. 3–12.

[45] C. J. D'Orazio, K.-K. R. Choo, and L. T. Yang, "Data Exfiltration From Internet of Things Devices: iOS Devices as Case Studies," *IEEE Internet of Things Journal*, vol. 4, no. 2, pp. 524–535, 2017.

[46] A. Nadler, A. Aminov, and A. Shabtai, "Detection of malicious and low throughput data exfiltration over the DNS protocol," *Computers & Security*, vol. 80, pp. 36 – 53, 2019.

[47] M. Ge, X. Fu, N. Syed, Z. Baig, G. Teo, and A. Robles-Kelly, "Deep Learning-Based Intrusion Detection for IoT Networks," in *2019 IEEE 24th Pacific Rim International Symposium on Dependable Computing (PRDC)*, 2019, pp. 256–25 609.

[48] N. Koroniotis, N. Moustafa, E. Sitnikova, and B. Turnbull, "Towards the development of realistic botnet dataset in the Internet of Things for network forensic analytics: Bot-IoT dataset," *Future Generation Computer Systems*, vol. 100, pp. 779 – 796, 2019.

[49] P. Amar, "DET: Data Exfiltration Toolkit," [Online; https://github.com/PaulSec/DET], 2016, accessed on May 20, 2021.

[50] Y. Nativ, "PyExfil: A Python Package for Data Exfiltration," [Online; https://github.com/ytisf/PyExfil], 2019, accessed on May 20, 2021.

[51] C. Tafani-Dereeper, "IPv6teal: Stealthy data exfiltration via IPv6 covert channel," [Online; https://github.com/christophetd/IPv6teal], 2019, accessed on May 20, 2021.

[52] SANS Internet Storm Center, "IPv6DNSExfil: Data Exfiltration and Command Execution via AAAA Records," [Online; https://github.com/DShield-ISC/IPv6DNSExfil], 2016, accessed on May 20, 2021.

[53] Arno0x, "DNSExfiltrator: Data exfiltration over DNS request covert channel," [Online; https://github.com/Arno0x/DNSExfiltrator], 2017, accessed on May 20, 2021.

[54] g0dmode, "dnsteal: DNS Exfiltration tool for stealthily sending files over DNS requests," [Online; https://github.com/m57/dnsteal], 2015, accessed on May 20, 2021.

[55] D. S. E. Deering and B. Hinden, "Internet Protocol, Version 6 (IPv6) Specification," Internet Engineering Task Force (IETF), RFC 8200, July 2017, accessed on Nov. 23, 2021. [Online]. Available: https://www.rfc-editor.org/rfc/rfc8200.txt

[56] RME-DisCo Research Group, "Chiton: Data exfiltration tool for IoT environments," [Online; https://github.com/reverseame/chiton], 2021, accessed on May 20, 2021.

[57] H. Shaban, "T-Mobile says hackers stole data of more than 40 million people," [Online; https://www.washingtonpost.com/business/2021/08/18/t-mobile-data-breach-hackers/], accessed on August 26, 2021.

[58] D. Shoemaker, A. Kohnke, and K. Sigler, *The Cybersecurity Body of Knowledge: The ACM/IEEE/AIS/IFIP Recommendations for a Complete Curriculum in Cybersecurity*. CRC Press, 2020.

[59] S. Rose, O. Borchert, S. Mitchell, and S. Connelly, "Zero Trust Architecture," National Institute of Standards and Technology (NIST), techreport NIST Special Publication 800-207, 2020.

[60] K. Scarfone and P. Mell, "Guide to Intrusion Detection and Prevention Systems (IDPS): Recommendations of the National Institute of Standards and Technology," National Institute of Standards and Technology (NIST), techreport NIST Special Publication 800-94, 2007.

Daniel Uroz received the BSc. in Informatics Engineering from the University of Zaragoza in 2016 and the MSc. in Cybersecurity Research from the University of León in 2020. Currently, he is pursuing a doctorate in Computer Science from the University of Zaragoza. His research focuses on the analysis of network protocols with formal models, with the aim of improving network defense solutions.

Ricardo J. Rodríguez received MSc. and PhD. degrees in Computer Science from the University of Zaragoza, Spain, in 2010 and 2013, respectively. His PhD. dissertation was focused on performance analysis and resource optimization in critical systems, with special interest in Petri net modeling techniques. He was a Visiting Researcher with the School of Computer Science and Informatics, Cardiff University, Cardiff, U.K., in 2011 and 2012, and the School of Innovation, Design and Engineering, Mälardalen University, Västeras, Sweden, in 2014. He was also a Visiting Professor at the University of Campania "Luigi Vanvitelli", Italy, during two three-month periods in 2016 and in 2018. He is currently an Associate Professor at the University of Zaragoza, Spain. His research interests include performability analysis, program binary analysis, and memory forensics. He has been involved in reviewing tasks for international conferences and journals.