CMXsafe: A Proxy Layer for Securing Internet-of-Things Communications

Jorge David de Hoz Diego, *Member, IEEE*, Taous Madi, *Member, IEEE*, and Charalambos Konstantinou, *Senior Member, IEEE*

***Abstract*— Security in Internet-of-Things (IoT) environments has become a major concern. This is partly due to a large number of remotely exploitable IoT vulnerabilities in service authentication and access control combined with the lack of timely technical support. To reduce the threat surface of remote vulnerability exploitation, we propose CMXsafe, a secure-by-design application-agnostic proxy layer that can be updated and managed independently of the IoT device application. CMXsafe places IoT devices behind gateways operating as 4th OSI transport layer relayers to offload security concerns of IoT network communications into the proxy layer. Specifically, the proxy layer produces secure communication paths between IoT applications and platforms while enforcing mutual authentication and access control to proxied services. We evaluate the performance of our architecture on the MQTT protocol used in a standard publisher-broker-subscriber configuration provided by Eclipse Mosquitto. We compare the performance penalty on the protocol when securing communications with TLS following a monolithic implementation and with CMXsafe. The experimental results suggest that CMXsafe outperforms integrated security by providing at least a 25% latency reduction and a 22% bandwidth improvement.**

***Index Terms*— Internet-of-Things, secure communications, socket proxy, secure proxy session, security context.**

I. Introduction

MOBILE wireless technologies support different types of Internet-of-Things (IoT) connectivity requirements, including massive machine-type communication. As a result, the number of IoT-connected low-cost devices is expected to rise over the coming years and reach 30.2 billion by 2027 [1]. Indeed, the harsh time-to-market race, along with the limited technical support, leaves the IoT devices and their Internet-exposed services marred by a wide variety of logical bugs (e.g., weak authentication and unauthorized access), software vulnerabilities, and poor default configuration settings [2], [3], [4], [5], [6]. As a matter of fact, over 60% of the IoT device Common Vulnerabilities and Exposures (CVE) documented during the last half of 2022 were remotely exploitable [7]. The compromised devices are then weaponized for lateral movement and for orchestrating Distributed Denial of Service (DDoS) attacks [8].

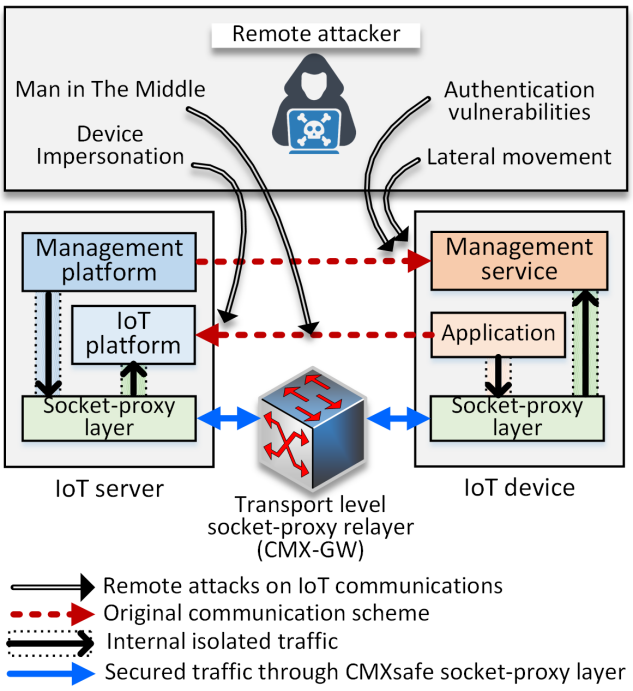

Fig. 1. IoT communications in traditional settings versus CMXsafe socket proxy layer to defend against remote attacks.

Other IoT security challenges are related to the limited support of secure communication protocols over the extended devices' lifespan, primarily because of their tight coupling with the core applications. This makes the firmware update a daunting and risky task [9], [10]. For instance, according to a recent investigation, even critical vulnerabilities in OpenSSL libraries have very long periods of patching delay (≈1454 days) in many IoT devices [11].

As illustrated in Fig. 1, most of the threats mentioned above are related to the Internet-exposed management and core IoT services (e.g., red arrows reflect the traditional communication paths between the IoT devices and the IoT platforms) and to the tight coupling of security protocols with the application logic. These two common security issues allow the following general types of remote attacks:

- Man-in-the-middle (MitM): These attacks can be motivated by outdated or incorrectly configured security

Manuscript received 4 December 2023; revised 7 April 2024; accepted 9 May 2024. Date of publication 22 May 2024; date of current version 29 May 2024. The associate editor coordinating the review of this manuscript and approving it for publication was Dr. Luca Caviglione. *(Corresponding author: Jorge David de Hoz Diego.)*

The authors are with the Computer, Electrical and Mathematical Science and Engineering Division, King Abdullah University of Science and Technology, Thuwal 23955, Saudi Arabia (e-mail: jdhozdiego@outlook.es; madileila@gmail.com; charalambos.konstantinou@kaust.edu.sa).

Digital Object Identifier 10.1109/TIFS.2024.3404258

TABLE I
COMPARISON BETWEEN EXISTING TECHNOLOGIES FOR SECURING THE COMMUNICATIONS OF IOT DEVICES APPLICATIONS

	Posix interface [12]	Network isolation [6], [13], [14]	Capture [15]	HALE-IoT [16]	CMXsafe
Malicious network access	✗	✓	✗	partial	✓
Support for security updates	✓	✗	✓	✓	✓

communication protocols in IoT devices, such as outdated or incorrectly validated server certificates.
- Device Impersonation: External attackers can leverage deficiencies in univocal device identification schemes that may allow attackers to use IoT devices with fake identities to generate a DoS and ultimately gain control over them.
- Authentication vulnerabilities: IoT device applications may expose outdated services with known vulnerabilities to the Internet, which can be detected remotely and exploited.
- Lateral movement: IoT devices operating in local networks with reduced security can be a target of attacks to gain access to other Information Technology (IT) systems and deploy further attacks.

As reported in Table I, existing solutions in the literature either address malicious network access (row one) or provide support for simplified security updates (row two). However, they fall short of addressing both concerns. For example, the frameworks proposed in [12] and [15] manage the third-party security libraries update or retrofitting security solutions into the firmware [16]. However, they do not provide countermeasures against malicious network access and lateral movement. On the contrary, in works like [6], [13], and [14], security gateways endowed with middleboxes (e.g., firewalls and intrusion detection systems) are proposed as a shim layer between IoT devices and the edge network. Those works address, to some extent, the network isolation and lateral movement issues, but they do not provide support for updating the security libraries. Hence, a fully-fledged application-agnostic solution for enforcing secure communication and simplifying security updates in communication protocols is still missing.

To address this gap, we propose in this work a Communication MatriX for an IoT safe (CMXsafe), a secure-by-design communication architecture with three main goals. First, we aim to limit the IoT device's service exposure (e.g., device authentication, service authentication, and authorization) by proxying communications after mutual authentication and applying access control to proxied services. Second, we set out to minimize the risk of infected devices' lateral movement by providing secure communication paths. Finally, we simplify the security protocols' patching process by leveraging the decoupling security concept in communication protocols [17].

To achieve these goals, CMXsafe offloads network communication security features from IoT devices and platforms into a proxy layer. This layer is composed of a containerized set of application-agnostic socket proxies, namely, CMX-GateWays (CMX-GWs), located between the IoT devices and the IoT platforms (Fig. 1). Those socket proxies are extended to IoT devices and servers via agents, namely, CMX-agents. The communications between the CMX-agents and the CMX-GWs proceed through standalone security transport layer communication protocols (e.g., SSH) via *Secure Proxy Sessions* (SPSs). CMX-agents and the CMX-GWs leverage built-in network services included in the Operating System (OS) network framework to derive *Security Contexts* (SCs). This virtually segregates local traffic within the IoT devices, servers, and CMX-GWs to produce secure, end-to-end, pre-established communication paths. IoT services are then exposed within the CMX-GWs solely through socket proxies without being directly interfaced with the Internet, and the secure communication paths are articulated within the CMX-GWs according to a set of predefined permissions.

This communication scheme takes advantage of the network traffic predictability of IoT systems. Contrary to other IT devices, many IoT systems perform specific functions that generate a reduced set of possible communication patterns. CMXsafe proposes blocking any direct communication with the IoT devices and centralizing services' authentication and authorization with predefined permissions. At the same time, the proxy layer allows a general standard procedure to offload security features from IoT device applications and platforms. We summarize the contributions of this work as follows:

- We present CMXsafe, a secure-by-design application-agnostic communication scheme for IoT deployments describing a novel proxy shim layer. We leverage the socket proxy-based SPSs and the OS-driven SCs concepts as generic local interfaces to devise end-to-end secure communication paths for IoT applications. The communication paths are articulated within the CMX-GWs, featured as 4th OSI layer relayers.
- We provide a security analysis of common vulnerabilities exploited by different attacks that can be averted or mitigated with CMXsafe. We describe the operation of each attack and the particular aspects of CMXsafe that take a relevant role against it.
- We provide an implementation of CMXsafe for a generic MQTT-based IoT deployment comprising 50 independent publishers, a broker, and a subscriber, all based on a standard Mosquitto MQTT embodiment.
- We assess the performance impact of securing the communications in terms of overhead, bandwidth, and latency. CMXsafe secured proxied communications are compared to traditional integrated TLS-based security.

II. CMXSAFE ARCHITECTURE

In this section, we discuss the intricacies of our architecture. First, we provide a high-level overview of CMXsafe with its implementation constraints. Then, we summarize its configuration and operation aspects, followed by the considered threat model. Afterward, we discuss the security properties of CMXsafe. Finally, we detail the communication model.

A. Overview

Fig. 2 presents a high-level overview of CMXsafe. In this architecture, all communications between the IoT devices and the IoT servers transit solely through a socket proxy layer.

Communications between IoT devices and IoT platforms are facilitated through the 4th OSI layer relayers, namely, CMX-GWs running on Commercial Off-The-Shef servers (COTS), where they can be deployed in container clusters. This proxy layer extends to the IoT devices and servers through the CMX-agents, simplifying mutual authentication. This facilitates holistic management of proxied communication between the IoT device applications and IoT platform services.

The CMX-GWs host a socket proxy server. This software accepts requests for SPS establishment (Section II-E1) from the CMX-agents and builds *secure communication paths* between the IoT device applications and the IoT platforms (Section II-E3). The CMX-agents run a socket proxy client to initiate requests for SPS establishment with the CMX-GWs. Both CMX-GWs and CMX-agents are configured by the CMX orchestrator, which is in charge of translating the service access permissions defined by the owner of the IoT devices into proxy configurations and communication policy rules. The CMX-GWs and CMX-agents enforce those policy rules as SCs. This allows the isolation of local inter-process communications (Section II-E2) and enforces access control. CMXsafe architecture is devised to simplify its implementation by abiding by the following constraints:

- The architecture must not modify the source code of the IoT device applications or the OS of IoT devices, gateways, or servers. The required functionalities should be standard, widely available, with active support, and thoroughly tested.
- The architecture requires a secure communication protocol with socket proxy features, but it must not be modified to guarantee compatibility and availability in IoT devices.

The following section describes the general procedures to configure and operate CMXsafe in an IoT deployment.

B. CMXsafe Configuration and Operation

Here, we describe the configuration and operation actions related to the setup of CMXsafe.

1) IoT Devices and Server Agent Bootstrapping: IoT device owners must claim ownership of their devices through the orchestrator of the IoT platforms. If the manufacturer of the IoT devices and the IoT platforms are the same stakeholder, the ownership claim process is straightforward through a unique registration key the owner receives with each IoT device. Otherwise, the IoT devices should be retrofitted with the CMX-agent provided by the orchestrator and receive the CMXsafe credentials through a provisioning protocol [18]. Likewise, the owner of the IoT platforms will allocate an account in the IoT servers to deploy the CMXsafe agent [19]. The orchestrator location depends on the specifics of each IoT deployment [20]. Business-critical environments can be located on the local site, whereas in distributed IoT environments, the orchestrator can be deployed as a cloud service.

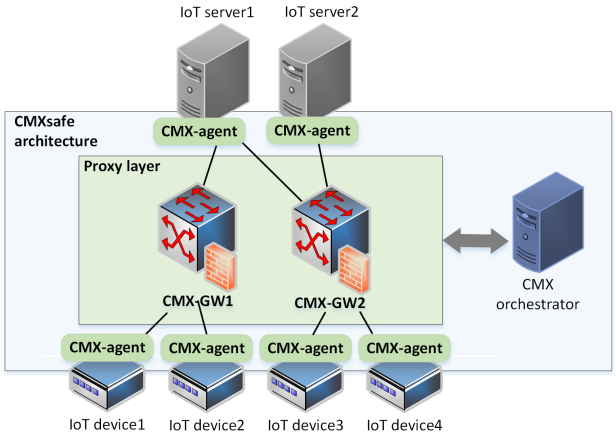

Fig. 2. High-level overview of CMXsafe architecture. CMXsafe proxy layer is composed of containerizable gateways and software agents that provide secure communication paths between IoT parties at the transport level.

Cloud IoT platforms are being progressively implemented as microservices abstracted from the underlying hardware (servers) [21]. In this case, CMXsafe implementation on the IoT platform side is facilitated thanks to a direct mapping of the microservice sidecar pattern [22], [23] with the CMX-safe server agents' operation. Likewise, as the CMX-GWs are stateless containers, they can be deployed as microservices, and the existing microservice mesh orchestrator can be extended to play as well the role of the CMXsafe orchestrator.

2) Communication Permissions: Once the IoT devices have been linked to their owners, they can configure the permissions to the available services offered by other devices and IoT platforms linked to the orchestrator. CMXsafe requires explicit permission to allow communications in CMX-GWs, e.g., permission of an IoT device to communicate with an IoT platform service. For example, a particular brand of IoT devices has permission to communicate with the service provided by the manufacturer's IoT platform. Thereafter, the orchestrator evaluates this permission and, once validated, translates it into SCs enforced at the CMX-GWs. This allows selectively revoking SCs if a device is compromised.

3) CMXsafe Operation: After the initialization phase, IoT devices and servers can establish communications through the CMX-GWs without the intervention of the orchestrator. This applies to most of the function-oriented IoT devices, as the communication between IoT devices and platforms relates to the function they provide, which usually remains unchanged during the device's lifetime. The CMX-agents from the IoT devices or servers can connect and disconnect from the CMX-GW without requiring further configurations, and the CMX-GW operates statelessly and autonomously according to the configuration provided.

Security updates deployed by the orchestrator that may affect the CMX-agents and their configuration will leverage the secure protocol proxying features. These updates include orchestrator commands for modifying the configurations in the event of access control permission changes or logical network topology modifications (e.g., IoT devices joining and leaving the architecture or ownership change). SC violation attempts

can also be detected in runtime, and this information can be leveraged for early detection of malfunction or attacks in IoT device networks to take action [24]. In case of CMX-GW failure, the situation can be automatically addressed, spawning a new one or adequately scaling the setup leveraging inherent resilience features available in cloud containerized clusters and microservice meshes.

C. Threat Model

Our architecture considers IoT devices with predictable network traffic. This is a common feature in IoT devices designed to provide specific functionalities, e.g., smart IoT meters recording energy consumption and communicating with the power utility server platform, surveillance IoT webcams communicating to the corresponding footage-storing server, etc. [25]. Contrary to multipurpose IT systems, the manufacturers of these types of IoT devices are able to describe the network services and protocols the IoT devices use to allow network traffic predictability [26]. This will facilitate cybersecurity certifications required by emerging regulations [27].

We assume that manufacturers will integrate the CMX-agent into their firmware. This action enables offloading security features into the CMX-agent, facilitating homogeneous and more inexpensive long-term security and communication management across different IoT devices. CMX-agent implementation in the IoT servers is feasible as they can be considered proxy middleware that does not modify the application protocols. We assume that CMX-GWs are run on trusted COTS. We also assume that the orchestrator is owned by the same stakeholder that manages the IoT platforms. This is reasonable when considering IoT platforms deployed in a microservice mesh. As the stakeholder devises more services, those can be managed as different IoT platforms through the cloud microservice mesh orchestrator. Communication with IoT platforms of different stakeholders/orchestrators is left out of the scope of this paper.

To allow a trusted architecture initialization, the manufacturer of the IoT devices should provide them with the CMX-agent. Otherwise, the IoT devices should be retrofitted with the CMX agent and securely bootstrapped. For this, the CMXsafe proxy layer relies on a trusted system, the orchestrator, that is capable of securely bootstrapping the CMX-agents of IoT devices and servers, providing them with CMXsafe credentials, and configuring and deploying CMX-GWs.

The way the CMXsafe orchestrator offloads the security of IoT platforms can be implemented following a sidecar pattern in microservice meshes, which is particularly useful when IoT platforms are already operating as microservices. In this case, the CMX-GWs and orchestrator can occur as other microservices integrated into the existing microservice mesh control plane. This facilitates CMXsafe deployment while benefitting from microservice meshes' flexibility, resilience, and high availability. Nevertheless, the scope of this paper comprises only the proxy layer, leaving the analysis of possible orchestrator implementations for future works.

Similarly to [15], we consider network-based adversaries trying to remotely compromise IoT devices by exploiting vulnerabilities in default services lurking in common third-party libraries. We also assume that some compromised devices in the IoT deployment might be trying to perform lateral movement to weaponize other devices or trying to access exposed services illicitly. We also consider attacks attempting to impersonate legitimate IoT devices through identity spoofing and MiTM attacks exploiting the lack of mutual authentication and certificate validation [2].

Considering the presented conditions, we elaborate in the next part on the security properties that CMXsafe will pursue and how they can be attained through the CMXsafe Secure Communication Model used to implement the proxy layer.

D. Security Properties

The design choices of CMXsafe aim to provide three security properties to significantly reduce the network-related threat surface: mutual authentication, centralized fine-grained access control management, and secure communication paths pre-establishment. This section discusses these security properties, how the decoupling of security in communication is preserved, and the implications of our design choices.

1) Mutual Authentication (P1): To provide full mutual authentication, the IoT server and IoT device must authenticate each other. However, this is frequently disregarded despite being essential to avoid device impersonation attacks [28], [29], [30]. Different solutions have been proposed to address device authentication. In [31], the authors propose a simplified key-sharing protocol for IoT device communications that generates different keys per device. In [32], blockchain technology is suggested to leverage hash chains for secure key management.

These technologies rely on algorithms that generate secure session keys for communication between devices or require intermediate nodes that act as information brokers to reduce the number of required keys. In CMXsafe, communications between peers occur within the CMX-GW. This requires IoT devices and servers to mutually authenticate *only* with the CMX-GW, significantly reducing the key exposure without requiring specific information brokers.

2) Centralized Service Authentication and Authorization (P2): Authentication vulnerabilities in Internet-facing services constitute a large threat surface that may result in full control of victims' IoT devices with IoT malware, such as Mirai [8]. This is particularly relevant in those without active technical support and disclosed vulnerabilities that can be located through IoT device scanners [33].

In CMXsafe, the IoT device and platform services are not exposed to the Internet, preventing undocumented and unauthenticated/poorly authenticated default services from being directly accessed. In fact, only services that are meant to be exposed can have their connection proxied to the CMX-GW according to pre-established permissions defined by the IoT device owner at the orchestrator level. These permissions are then translated into secure communication paths enforced with SPSs and SCs.

3) Secure Communication Path Establishment (P3): The vulnerabilities in IoT device services can lead to lateral movement attacks to compromise other IoT devices through remote

attacks. Some of those attacks might be stealthy and can go undetected when their generated traffic fits into accepted categories [34], [35]. To address those attacks, in CMXsafe, the services forwarded to the CMX-GW are only accessible by devices and platforms that are previously authenticated and have enabling SCs for communications, limiting the risk of anonymous fuzzy/injection attacks [36]. Lateral movement within the CMX-GW is unfeasible without compromising it because the access control to proxied services enforced with SCs prevents an authenticated device from connecting to those it has not been authorized. SCs also allow immediate detection of unauthorized communication attempts, enabling early identification of compromised devices and mitigation actions [24]. The entire CMX-GW can be statelessly containerized and managed in the cloud, difficulting compromising attacks thanks to available cloud technologies applicable at the orchestrator level. This contributes to reducing the surface attack compared to the one usually present in edge gateways and also increases scalability, availability, and resiliency [37].

E. Secure Communication Model

In CMXsafe, we propose to handle all IoT device communications through a proxy layer composed of a set of CMX-GWs. More specifically, communications between IoT devices and IoT servers transit solely through pre-established *secure communication paths* defined by two types of segments: a) *SPSs* (Section II-E1) for remote communications between the CMX-agents of IoT devices and servers with the CMX-GWs. b) *SCs* (Section II-E2) for local inter-process communication within the IoT devices, servers and the CMX-GWs.

This communication scheme leverages privilege separation features provided through OS UAs to provide IoT deployments with the security properties presented earlier. As illustrated in Fig. 3, we identify the following types of UAs:

- The IoT device UA and the server UA are meant to host/run the IoT applications and platforms. They serve as a differentiating feature for SCs to determine which local sockets are accessible from processes run under these UAs.
- CMX-UAs inside the IoT devices/servers trigger the establishment of SPSs with the CMX-GWs using public key authentication.
- The IoT device/server UAs in CMX-GWs serve as anchor points to propagate the identity of IoT devices/servers inside the CMX-GWs when they authenticate into the CMX-GWs.

A secure communication path is established when traffic from one UA within the CMX-GW is allowed to access a service proxied within another UA in the CMX-GW. In Fig. 3, the secure communication path allows the IoT device application to reach the Service's socket of the IoT platform. The following two sections detail the mechanisms underlying SPSs and the SCs that allow the composition of secure communication paths.

1) Secure Proxy Sessions: SPSs are established between the CMX-GWs, IoT devices, and servers before any network traffic is exchanged. To this end, the CMX-GW runs a Server Socket Proxy, while the CMX-agents within the IoT devices and IoT servers run Client Socket Proxies (CSP) to initiate the SPS establishment. The CMX-agents use their CMXsafe credentials, which consist of a public/private key pair, for authentication and secure session negotiation. Each IoT device has a unique CMX-UA wherein the CMX-agent is running, whereas the IoT server has a CMX-UA for each CMX-GW it is connected to with a dedicated CMX-agent. The CMX-GW authenticates the CSPs within the IoT devices/servers' respective UAs when they use their public authentication keys. This facilitates identifying later each SPS to the corresponding IoT device or server and allows propagating this differentiation to the generated proxies and traffic within the CMX-GW.

We identify two scenarios for secure communication path establishment depending on which party exposes the services whose connection needs to be forwarded through the SPS: a) IoT platform/server accessing a service exposed by an IoT device's application. In this case, the IoT device runs the server application process(es), and the b) IoT device accessing a service exposed by an IoT platform, running as a server application process. In both cases, we identify the following types of protected sockets to enable the service's communications to be end-to-end securely forwarded through the proxy layer:

- *Service/destination socket (Ser-sock):* It is defined at the server application process to listen to clients' requests.
- *Reverse Socket Proxies (RSP):* It is generated at the CMX-GW and refers to the exposed service at the server application side. In other terms, the RSP brings the service exposed either by the IoT device or the IoT server to the CMX-GW level as a listening socket to the corresponding requests, as if it were a copy of the service/destination socket.
- *Direct Socket Proxy (DSP):* It is established by the CSP running on the side of the client application process to forward local traffic to the RSP to access the remote service. In other terms, the DSP brings a remote service exposed inside the CMX-GW as a listening socket to the corresponding local requests.
- *Ephemeral sockets (Cli-sock):* In a communication between sockets, the one source of the communication request is defined as the ephemeral socket, which is terminated once the communication ends. The port number of the ephemeral socket is usually randomly assigned by the kernel.

CMXsafe assigns identities accounting for IPv6 local addresses for each device/server by combining one of its own MAC addresses with another one of the CMX-GW (i.e., [#MAC-GW|#MAC-DEV], where # is a known prefix). Contrary to traditional IoT implementations where device ID secrecy is required [38], in CMXsafe, the combination of these addresses with mutual authentication avoids device impersonation. To preserve the identity of devices in proxied communications, CMXsafe leverages the capacity of proxy mechanisms (i.e., SSH). This allows specifying the features of locally used IP addresses and ports, which are validated by

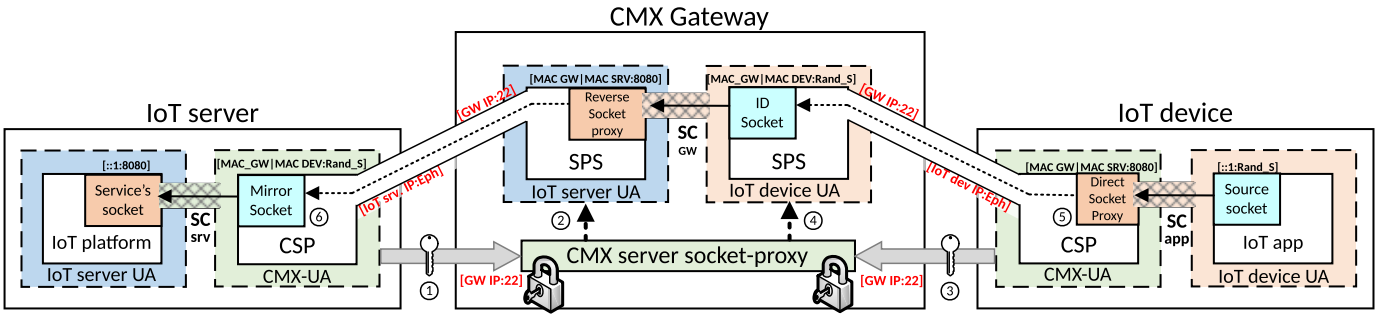

Fig. 3. Secure communication path established from the IoT device application to the IoT platform through CMXsafe.

the local SCs on runtime [24]. CMXsafe also determines the ephemeral sockets used in proxied communications:

- *Identity socket (ID-sock):* It replaces the ephemeral socket used as the source of requests to a proxied service. It is created by the SPS to preserve the identity of the service requester (the source) at the CMX-GW level. The identity is mapped into the source address of the proxied requests.
- *Mirror socket (Mir-sock):* In a similar way to the ID-sock, the Mir-sock is created by the CSP on the server application side to preserve the identity of the service requester. It is an identical copy of the ID-sock assigned at the CMX-GW.

In the next part, we discuss how SCs leverage identity preservation to enforce access control to sockets and how they isolate inter-process network traffic.

2) Security Contexts: SCs are OS-driven artifacts that rely on the concepts of network framework (e.g., Netfilter) and protected sockets [24]. They allow traffic from a specific UA to a particular local socket isolated from other local traffic within the IoT devices and servers. SCs are also applied in the internal CMX-GW communications between the SPSs of different UAs. SCs consider standard traffic features, i.e., source and destination IP addresses and destination ports. However, SCs can also classify the traffic according to the local UA running the process source of the traffic. This allows for enforcing access control to proxied services.

CMXsafe uses SCs to map a Connection ID (ConnID) to <UA, S>, where UA contains the process that constitutes the source of the traffic communication request, and S refers to a protected destination socket ($SC : UA \times S \rightarrow ConnID$). The ConnID is the source local IP address and uniquely defines the UA source of the communication thanks to the IPv6 mapping used by *ID-sock* and *Mir-sock*.

CMXsafe local IPv6 addresses are isolated by default and cannot be used to send or receive information unless an SC allows it. The isolation and the SCs are composed of marking and filtering traffic rules. These comprise three general rules for the entire device that isolate the communication sockets and one specific rule for each SC that allows fine-grain access control to each service, either local or remotely proxied [24].

3) Secure Communication Path: The combination of SPS and SCs allows the establishment of secure communication paths between the IoT device application and the IoT platform at the server. These secure communication paths preserve the identity of the source IoT device, allowing an authentication procedure at the server based on the device ID. To accomplish this, the source IP address of the incoming communication request is linked univocally to the source IoT device by the CMX-GW and preserved through the CSP in the server, which allows the IoT platform to reliably verify the source of each proxied packet received.

Furthermore, the secure communication paths prevent local impersonations thanks to the SCs, and any remote attack attempt would have first to bypass the security features provided by the proxy protocol (e.g., SSH). The combination of SPS with SCs enables the establishment of a secure communication path through which application communications are subsequently forwarded, secured, and isolated. Note that the same description applies when a service is exposed by an IoT device, except that the DSP is defined on the IoT server side.

The architectural constraints of the model to satisfy the security properties of CMXsafe can be verified with modeling languages and verification tools such as Alloy [39].

III. Security Analysis

CMXsafe pursues improving network isolation, securing communications at a transport protocol level, and preserving the network traffic's device identity (device ID). We have summarized this in Table II, including a brief description of the attack related to each vulnerability, its potential effects, and CMXsafe capabilities for prevention and mitigation.

1) Lack of (Reliable) Mutual Authentication: Deploying this type of authentication in IoT environments is challenging because clients must present unique credentials that are trusted in the API request [40], but using a single certificate for all IoT devices can be problematic [41]. Here are some attacks related to inadequate authentication/certificate validation:

IoT Device Impersonation: An attacker replaces the victim's IoT device with a fake device, i.e., a software entity that resembles a real IoT device to the IoT platform. This fake device uses the ID of the real device to impersonate. The root cause for this vulnerability is that the IoT platform cannot differentiate a legitimate device ID from a counterfeit ID. This vulnerability allows the attacker to remotely hijack the victim's IoT device [29], [30].

In CMXsafe devices, a single secure communication path provided by proxies is linked to the device ID. It allows communication between the IoT device application and IoT platform service through the CMX-GW. An attacker would require gaining the public/private keys of the victim's IoT

TABLE II
SUMMARY OF COMMON VULNERABILITIES ADDRESSED BY CMXSAFE

Vulnerability	Attack	Potential effect	CMXsafe mitigation
Lack of (reliable) mutual authentication or certificate validation	Remote device impersonation [29] [30]	The victim's IoT device is controlled by a remote attacker using a registered device ID matching the victim's IoT device	Once permissions are changed, the orchestrator terminates the corresponding security contexts, and the device or server can no longer access them
	MiTM [2]	The attacker can eavesdrop/alter all communications	IoT platform authentication through public key pinning to establish the proxy sessions
Lack of service isolation	Device authentication bypass [42]	The attacker can log into IoT device administrative services	Access to services only occur through CMX-GWs internal proxies where access control is applied according to communication policies enforced with security contexts
	Service vulnerability exploitation [43]	Remote code execution	
	Lateral movement [34] [35]	Attack escalation across DMZ	
Outdated security/configuration in communication protocols	Whole key recovery [44] [45]	Privacy loss	Security for communication protocols is decoupled from the IoT device applications and can be timely updated without affecting the rest of the software

device to successfully render this attack, which makes it challenging as it would require first physical access or tampering with the victim's IoT device. CMXsafe architecture also facilitates the application protocol layer to perform cross-layer identity verification (Section IV). As an example, this allows an IoT platform using MQTT to detect any attempt of device ID impersonation by an IoT platform user [2] because of the mismatch between the claimed IoT device ID at the MQTT protocol and the one imprinted in the source address of each received proxied packet through the secure communication path.

Man In the Middle Attacks (MiTM): Many IoT devices trust IoT platform certificates without validating their signature with a Certificate Authority, which allows MiTM attacks [2]. In CMXsafe, the CMX-GW is authenticated by the fingerprint of its public key by the CMX agent. This allows the IoT devices to validate the identity of the CMX-GW before establishing the SPS with the CMX-GW. Thus, independently of the destination, any following proxied communication through the SPS does not require certificate validation because all other SPS established to the CMX-GW have also been authenticated likewise.

2) Lack of Service Isolation/Effective Access Control: IoT devices are usually remotely managed through the IoT platform. However, some devices also allow services for direct access through the network, which may be used as an entry point for remote attacks:

Device Authentication Bypass/Service Vulnerability Exploitation: These attacks affect IoT devices with a mini Web server or a customized application server with a listening port that allows users to access and control the device directly. Weak credentials or vulnerabilities in the service allow remote exploitation [42], [43].

Lateral Movement: The compromised device leverages the reduced security in DMZs where the IoT device operates to deploy lateral movement attacks to compromise other IoT devices or IT systems. Although this type of attack can be successfully identified and mitigated in many scenarios, stealthy attacks can go undetected when they fit into accepted traffic categories [34], [35].

CMXsafe limits these threats, reducing the attack surface by only proxying the required services into the CMX-GWs. The rest remain unreachable even in the local network. Furthermore, proxied services at the CMX-GW are only accessible by other devices or platforms authorized for that through SCs, further limiting the attack surface and preventing anonymous fuzzy/injection attacks (Section II). Lateral movement within the CMX-GW requires compromising the CMX-GW because the SCs prevent an authenticated device from connecting to unauthorized proxied services.

3) Outdated Security in Communication Protocols: IoT device application protocols are prone to operate outdated or inadequate configurations for TLS eventually [46]. This enables different attacks related to outdated/vulnerable cipher suites configurations [47], or discovered vulnerable keys in certificates [45].

Whole Key Recovery Attack. As a result, the traffic from IoT devices using outdated security implementations for communications is subject to efficient key recovery attacks [44], [45]. The difficulty of upgrading the firmware, together with the extended lifespan of the devices, motivates this situation.

CMXsafe decouples security communication protocols from IoT device applications, facilitating their reconfiguration or updating as source code independence is preserved.

A. Potential Security Issues

In this part, we describe potential security issues on the servers, CMX-GWs, and orchestrator and assess their impact.

1) IoT Server: IoT platforms hosted in IoT servers comprise the front end of IoT deployments interacting with end-users and external IT systems. This causes the attack surface in IoT servers to be larger than in the CMX-GWs or the orchestrator, increasing the possibility of successful intrusions [48], [49]. If an IoT server becomes compromised, the data provided by the IoT platforms can be altered, and the services depending on it may become affected. Nevertheless, the key credentials of IoT devices are not compromised as secure communication paths are established in the IoT back-end of the deployment within CMX-GWs.

2) CMX-GWs: The CMX-GWs are less exposed to attacks than IoT servers due to the limited types of traffic they process and the software they run, substantially reducing the attack surface compared to IoT servers. Nevertheless, we can assume that a CMX-GW has been compromised. In this case, the communications that the CMX-GW handles are exposed, and confidentiality and integrity are lost. This means that the CMX-GW could also impersonate any IoT devices it manages but not those from other CMX-GWs. The CMX-agent in the server would reject the communications because there is no SC

to allow them from the CPS of the compromised CMX-GW. It is worth noticing that the CMXsafe proxy layer structure would also facilitate the outlier detection of the compromised CMX-GW at a network level after any lateral movement attempts out of secure communication paths.

3) Orchestrator: If the orchestrator is compromised, an attacker could leverage it to deploy arbitrary software with the CMX-agents. After restoring the orchestrator to an uncompromised state, IoT devices should be reset to factory defaults, and all CMX-agents should be replaced and provided with fresh credentials. The integrity of the server is preserved as long as the privilege separation of the UAs withstands any potential code execution by the attacker. Without privilege escalation, lateral movement from the assigned UAs to CMXsafe in the server is impossible. Orchestrator vulnerabilities can be caused by default configurations that foster compatibility and interoperability at the expense of security or design flaws in the microservice mesh [50]. Thus, the CMXsafe proxy layer contributes to mitigating limitations on microservice meshes' security mechanisms, providing security properties that reduce the attack surface. Furthermore, CMXsafe deployments can leverage a runtime verification of security policies thanks to the use of SCs [24], an uncovered gap that can significantly contribute to limiting the impact of attacks and accelerate mitigation actions [51].

IV. Implementation and Evaluation

CMXsafe pursues the security properties described in Section II-D (*P1, P2, P3*) while abiding by the implementation constraints described in Section II-A Thus, the proposed implementation, according to Fig. 3 is as follows:

- **The SCs** can be deployed using existing functionalities of a generic network framework able to classify traffic according to the OS UA source of it.
- **The proxy protocol** required by CMXsafe uses features present in SSH, a thoroughly tested protocol with multiple embodiments and active support. The server socket proxy and CSP can be provided as an SSH server and client configured to allow only socket proxy functionalities as the DSP and RSP are features available in the SSH protocol. The SSH client used as CSP in the IoT devices can be any of the available as long as it supports key-based authentication and port forwarding. In fact, proxy capabilities can be achieved with other client/server proxies supporting port forwarding based on different security protocols, such as TLS or DTLS. Nevertheless, the motivation to evaluate SSH is its wide implementation in production servers and gateways and its increasing availability on constrained IoT devices and RTOS. This allows a considerable maturity level thanks to active development due to its constant exposure to threats. TLS and DTLS are also widely used and mature but as integrated security solutions on applications, not as stand-alone proxies. Socket proxy capabilities are functionalities present in SSH specification [52] but not in TLS/DTLS.
- **CMX-agents** implementations in IoT devices comprise a proxy protocol portable client (SSH portable client) as the CSP, its configuration, and scripts for initializing and updating. The footprint of the proxy client varies according to the embodiment chosen. For example, the WolfSSH client has minimum requirements between 1.4KB and 2KB of RAM and 33KB of ROM and provides active support for different chipmakers and RTOS [53]. The integration of the CMX-agent is straightforward in Linux-like systems with POSIX features and an Application Binary Interface (ABI), which also applies to different legacy Linux devices [16]. This can be done by customizing the initramfs, which runs the CMX-agent during the initial booting stages. Constrained IoT devices running RTOS like Microsoft Azure ThreadX [54] can also include CMX-agents as tasks dynamically loaded as external precompiled modules. Using other RTOS is also feasible if source code independence with the main application/thread is preserved. Nevertheless, in this case, the entire firmware must be linked again after the agent update.
- **Device ID management** for proxied communications is based on MAC and IPv6 addresses. Real traffic between IoT devices, gateways, and servers is independent of this virtual addressing for proxied communications. This allows a decentralized assignation of virtual addresses while minimizing the risk of collision in case different IoT CMXsafe-based deployments decide to merge.
- **The CMX-GW** can be implemented as a stateless containerized service, simplifying its management, deployment, and remote attestation. This implementation can take advantage of SSH proxy protocol, its existing OpenSSH embodiment, and standard OS and socket API features:

(a) OpenSSH leverages OS privilege separation features to manage each SPS in different UAs based on the credentials and keys used.

(b) The Linux UAs of the IoT devices and servers inside the CMX-GW are assigned a unique virtual IPv6 address within the CMX-GW. The length of a fully extended IPv6 address is 32 characters (once the colons are removed). As the default maximum Linux UA name length is usually 32 characters, the IPv6 address mapping to UAs can be made by setting the virtual IPv6 address as the UA name. Then, the SPS (Fig. 3) can retrieve the assigned IPv6 address by looking into the UA it has been deployed. This speeds up proxied connections managed by the CMX-GW, as no database requests or domain resolutions are required.

- **The ID-sock** is created by the SPS when forwarding a communication request of the IoT device to an available proxied service within the CMX-GW. The ID-sock imprints the device's assigned virtual IPv6 address into the traffic to the proxied service. This is done by populating that info in the *sockaddr_in6* structure before the *bind()* and *connect()* operations to communicate to the proxied service.
- **The Mir-sock** is generated in the IoT server by the CSP. It uses the source IP address of the proxied communication for the ID-sock imprinted in the CMX-GW. This information is available in the proxy channels and can be passed to the *sockaddr_in6* structure before *bind()* and *connect()* operations.

A. Benchmarks Configuration

In this section, we present a test series on a testbed considering different scenarios to assess the performance of

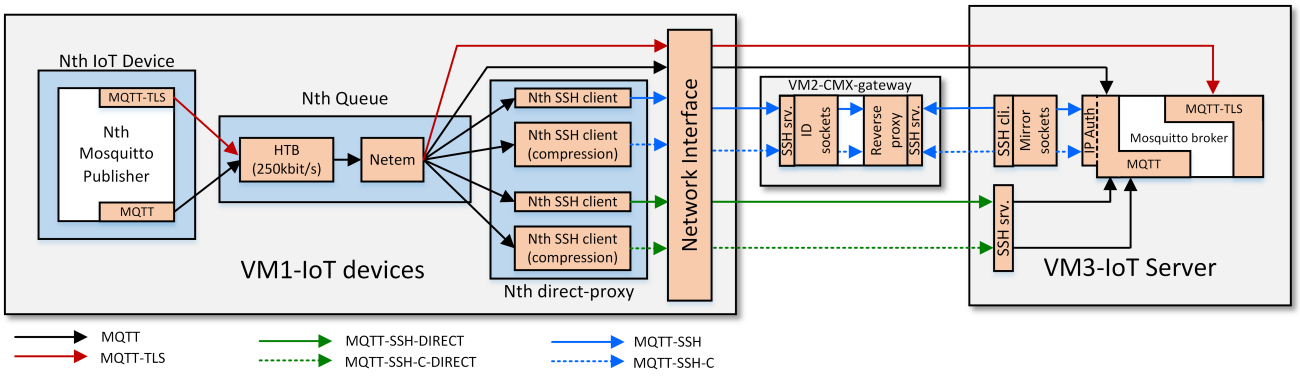

Fig. 4. Testbed layout to compare the performance of different configurations in securing MQTT communications.

CMXsafe securing IoT traffic when compared to default integrated TLS-based securing techniques. Fig. 4 depicts a testbed where IoT devices publish messages into a particular topic of an MQTT broker in an IoT server. The information is then delegated to another subscriber (not included in the figure). We analyze the MQTT protocol as it is widely used in industrial IoT environments [55] but also frequently insecurely deployed [33]. The testbed implements the major elements of the CMXsafe proxy layer as follows:

- OpenSSH standalone 9.2p1 clients for the IoT devices supporting port forwarding and key authentication [56].
- A customized embodiment of the OpenSSH server and client implementing the ID and mirror sockets on the CMX-GW and server without modifying the SSH protocol.
- A direct mapping between IPv6 addresses and CMX-GW UAs used by devices and the server.
- MQTT Mosquitto V2.0.15 publisher-broker-subscriber [57].
- MQTT authentication by IP leveraging a Mosquitto plugin and CMXsafe features.

The testbed configuration considers 50 different IoT devices publishing sequential messages to the broker. Each device is represented by a publisher whose traffic is shaped by capacity and scenarios relevant to constrained IoT devices, such as low-powered wireless devices using 6LoWPAN [58]. To accomplish this, a Hierarchical Token Bucket (HTB) queue of 250 kbps is added to each publisher, followed by a Netem filter to add delay, jitter, and packet loss ratio according to each scenario. Different payload sizes are tested, comprising random text. The following scenarios are analyzed:

a) Ideal channel: 5 ms. delay, no jitter, no packet loss.
b) Real channel: 25 ms. delay, no jitter, no packet loss.
c) Lossy channel: 125 ms. delay, 5ms. jitter, 1% packet loss.

The following configurations are compared:

- [mqtt]: Plain MQTT with an anonymous identification.
- [mqtt-tls]: MQTT with TLS with anonymous identification.
- [mqtt-ssh-direct]: MQTT proxied through SSH directly to the server with anonymous identification.
- [mqtt-ssh]: MQTT proxied through SSH and the CMX-GW with device authentication.

SSH configurations are also considered with SSH compression features enabled (mqtt-ssh-c and mqtt-ssh-direct-c). These tests are set with the standard configuration options for MQTT publishers and brokers provided by the Mosquitto embodiment. In particular, QoS is set to 2 under MQTT V3. TLS configuration is set up using one of the cipher suites recommended [47] for TLS 1.2: TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384. SSH's current version does not include the exact algorithms used by the MQTT publishers. Thus, the options chosen for SSH are aes256-ctr as a cipher, hmac-sha2-512 as a MAC function, and sntrup761 × 25519-sha512 as an NTRU lattice-based key exchange algorithm.

The setup comprises three virtual machines powered by i7-1185g7 logical cores at 3GHz, each one running UBUNTU 18.4.06 LTS with a Linux Kernel v4.15:

- VM1 includes 50 MQTT publishers as IoT devices and the networking environment. Each MQTT client has its SC and CMX-agent to connect to the CMX-GW. VM1 runs with 3 logical cores and 3 GB of RAM.
- VM2 features the CMX-GW. There, each IoT device and the server have a UA and IP address according to CMXsafe architecture. VM2 uses 1 logical core and 3 GB of RAM.
- VM3 includes the MQTT broker and CMX-agent to connect to the CMX-GW and uses 2 logical cores and 3 GB of RAM.

The communications of the 50 publishers are set simultaneously in parallel, and the time required for each IoT device to perform each communication (Treq) is measured individually from within the MQTT publisher. We consider a general case where a total of 300 different publication messages are sent by each publisher sequentially (one publication operation after another), and the Treq is averaged. A particular corner case is also analyzed in which the 300 messages are sent at once in a single publish operation. This allows for assessing TLS performance when a single TLS session is used in different message transmissions. This is considered a corner case because it would require TLS session resumption, a feature not implemented in the MQTT mosquitto publisher.

B. Results

The outcome of the test series is presented in Fig. 5. The Treq used per publish operation is displayed for each case, together with the provided bandwidth ratio (BW) when compared to plain MQTT publish operation. Different payloads are analyzed, but we focus on lower ones, as many MQTT communications in low-powered IoT environments are frequently short text-based messages.

1) Treq Analysis Between mqtt-tls and mqtt-ssh: The first scenario outcome is depicted in Fig. 5a. The mqtt-ssh configuration for 64 bytes of payload requires a Treq 36% lower than mqtt-tls. The Treq difference fluctuates between 25% and 56% with different payloads and channel conditions, as in scenarios (b) and (c). In all these tests, the better performance of mqtt-ssh over mqtt-tls can be explained because TLS requires establishing a different secure session per publish operation. This involves opening TCP connections between the IoT devices and server, validating certificates, key exchange negotiations, etc. On the contrary, the mqtt-ssh configuration only requires proxying the plain MQTT publish requests through the local proxies.

2) Overhead and BW Analysis Between mqtt-tls and mqtt-ssh: When considering the overhead of different security configurations (Fig. 6), we have analyzed the absolute overhead as the ratio between "bytes sent" through the network interface and the mqtt payload (message sent). According to this definition, there is overhead when the ratio exceeds 1. This metric includes the SSH proxy protocol overhead related to each SSH communication channel establishment upon each proxy communication request [59]. The overhead measured also takes into consideration the related overhead of the involved protocols of other OSI layers simultaneously. The reason for not comparing the proxy's overhead with TLS independently is the correlation between all involved protocols when communications require packet/datagram fragmentation or retransmission handling and the impact of this on the absolute overhead.

Implementing security in communications consumes part of the maximum available BW, which is only attainable by plain mqtt and is considered a reference. The required BW by mqtt-tls to provide security in scenario (a), at low payloads (64 bytes), is 52% of the total BW available, whereas mqtt-ssh only requires 24%. This equals a 28% reduction in the required BW to secure the communications. These results are congruent with an overhead analysis. The overhead required by mqtt-tls to deploy security is 4.68 times more than mqtt-ssh at reduced payloads (64 bytes). As the payload increases, the bandwidth figures related to mqtt-tls improve as the extra overhead required during the TLS session negotiation has less weight on the complete publish operation. The reduction in the required BW is between 22% and 46%, depending on the payload and scenario considered.

3) Overhead and Fragmentation: Fig. 6 shows that mqtt-tls has a higher overhead ratio at low payload values. This is caused by the TLS session establishment not being experienced in CMXsafe-based communications due to its ability to leverage established secure communication paths. However, as payload increases (particularly between 1KB and 2KB), plain mqtt and non-compressed mqtt-ssh configurations have their overhead ratio progressively closer to mqtt-tls. In the testbed analyzed HTB filters contribute to the overhead by inducing the action of TCP congestion control mechanisms. Another source of overhead that applies generally is related to the required fragmentation of the increasing payload into multiple Ethernet frames due to the Maximum Transmission Unit limit of the physical layer frames (1.5 KB in Ethernet).

As the mqtt payload continues increasing (above 2KB), the overhead of non-TLS configurations grows even closer to the overhead produced in mqtt-tls. This has an asymptotic behavior at much larger payloads (payload$\gg$>4KBytes) towards an overhead ratio slightly above 1. This is expected, as communication protocols in stationary communication channels will incur a quasi-proportional overhead to large payloads once the session establishment overhead is no longer relevant. Compressed communications do not behave following this pattern on the analyzed tests as they are able to achieve overhead ratios below one because of the high compressibility of text-based payloads used.

4) Analysis of mqtt-ssh and mqtt-ssh-direct: The mqtt-ssh-direct configuration is considered to allow quantifying the penalty introduced by the CMX-GW operation. For 64 bytes of payload, the mqtt-ssh Treq is 23% greater in scenario (a). This penalty is reduced in higher payloads or when the channel properties worsen. In scenario (c), it is reduced to 7%. When compression features are enabled, this produces a slight overhead increase at low payloads that is greatly compensated when payloads increase (Fig. 6).

5) TCP Three-Way Handshake in Proxied Communications: Mqtt-ssh-direct has a slightly lower Treq than mqtt in scenario (c). This effect is due to the three-way handshake used in TCP when establishing a connection. While mqtt requires that handshake from the device to the server, mqtt-ssh-direct allows the mqtt publisher to do it locally in the VM1, as the proxied connection to the server is already established by SSH. This also occurs in the server, but the resulting Treq is still sufficiently small to compensate for the delay in encryption.

6) Corner Case Analysis: A major advantage of CMXsafe operation when compared to TLS is the availability of a secure communication path, which speeds up the publishing operations at the application protocol layer. TLS can provide session resumption, but as a general case, it is not considered. This feature is indeed available in TLS 1.2 "*at the potential cost of certain security properties*" [47], and in TLS 1.3, it is upgraded with the 0-RTT feature to resume sessions more securely and faster. Nevertheless, its implementation should be avoided "*unless an explicit specification exists for the application protocol in question to clarify when 0-RTT is appropriate and secure*" [47]. Neither the MQTT publisher nor the subscriber provided by Mosquitto support TLS session resumption, which made us consider it a corner case. We analyze this in Fig. 7 applying scenario (c). In this case, each publisher sends 300 messages in a single publish operation, and then Treq is averaged. This allows mqtt-tls to use the same session across different messages. At lower payloads (64 bytes), we can appreciate that the difference between mqtt-tls and mqtt-ssh with plain mqtt Treq is less

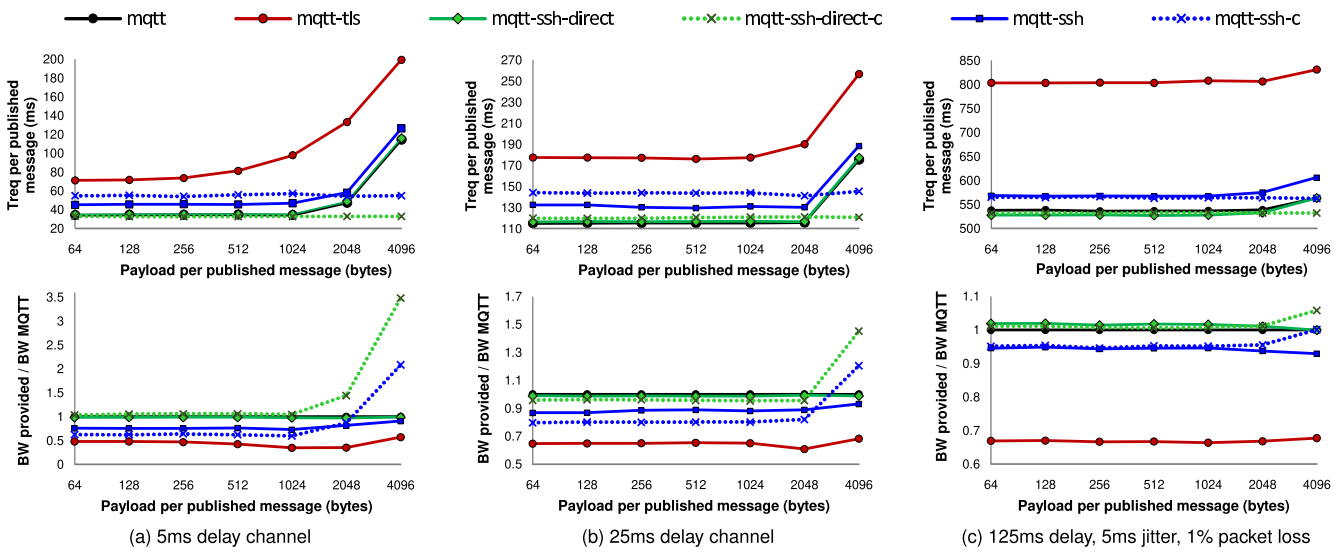

Fig. 5. Treq per published message and BW ratio provided by each configuration operating under three different scenarios (general case). The horizontal axes are in log scale.

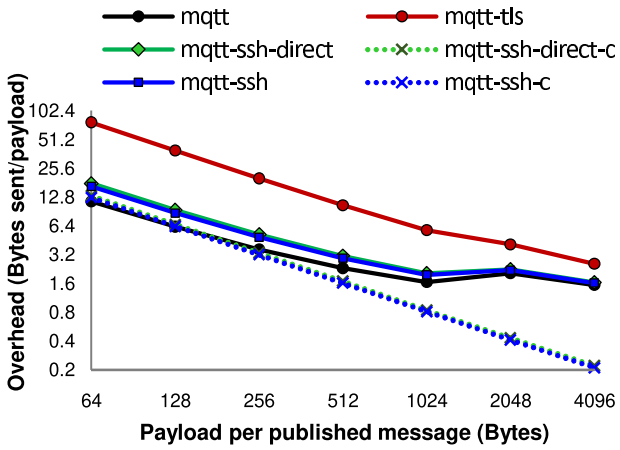

Fig. 6. Overhead analysis of security configurations on a 5ms delay channel (general case). Both axes are in log scale.

than 1.1%. As payloads increase, the mqtt-ssh Treq difference with mqtt increases to 4.5%, while the mqtt-tls Treq difference with mqtt is only 2.6%. This is due to the extra overhead mqtt-ssh requires to handle the proxy channels, which reduces the available bandwidth and translates into increasing the Treq when payloads increase. Nevertheless, this effect is easily mitigated when built-in compression features of SSH protocol are enabled, as depicted in Fig. 7 with the mqtt-ssh-c configuration.

V. Discussion

The results obtained in the test series are aligned with the results of other works that have evaluated the performance of SSH embodiments as proxies in real IoT devices (raspberries). Those tests considered real 802.15.4 interfaces and different types of traffic (HTTP, HTTP/2 CoAP) [17], [60]. The computation requirements and overall performance improved in proxied communications. However, the RAM and ROM footprint requirements are limiting factors and depend on the chosen proxy embodiment. For instance, the DropBear SSH client can compile to a 110 KB statically linked binary with uClibc [61]. Optimized SSH embodiments such as WolfSSH for constrained IoT devices can reduce ROM footprint to 35kb of ROM [53]. RAM consumption also depends heavily on the SSH embodiments, ranging between 2 and 7 MB for standard client versions (compiled with all default features) [17]. Nevertheless, optimized SSH clients only require between 1.4 KB and 2 KB of RAM plus the configurable receive buffer [53]. This facilitates the implementation of CMXsafe in constrained IoT devices comprising 50 KB of RAM and 250 KB of ROM (Class 2 devices), or 10 KB of RAM and 100 KB of ROM (Class 1 devices) [62]. Class 0 devices are considered highly constrained and unsuitable for directly interfacing with the Internet (as they are furnished with less than 10 KB of RAM and 100 KB of ROM). CMXsafe can provide a progressive implementation, and in this case, its support would focus on Internet-exposed systems only, i.e., the IoT gateways used in those types of deployments. These devices are usually categorized above Class 2 and frequently operate under Linux-based OS systems [55], which can implement the CMXsafe-agents straightforwardly, as SSH clients are already available on many of those gateways. CMXsafe-agents can leverage existing SSH clients and easily update them with portable stand-alone versions. In cases where the Linux implementation is reduced to a Busybox-based system [63] (a single Linux binary), it is still feasible to include the CMX-safe agent through the initramfs at the booting stages.

The applicability of CMXsafe in IoT deployments is preferred in those with IoT devices or edge devices with Linux-based systems or RTOS with POSIX support. According to a recent survey [55], 51% of edge IoT devices and 43% of constrained IoT devices already use Linux. Nevertheless, constrained IoT devices may struggle to include the CMX-safe agent. However, in these IoT systems (probably running RTOS), implementing the CMX-safe agent requires

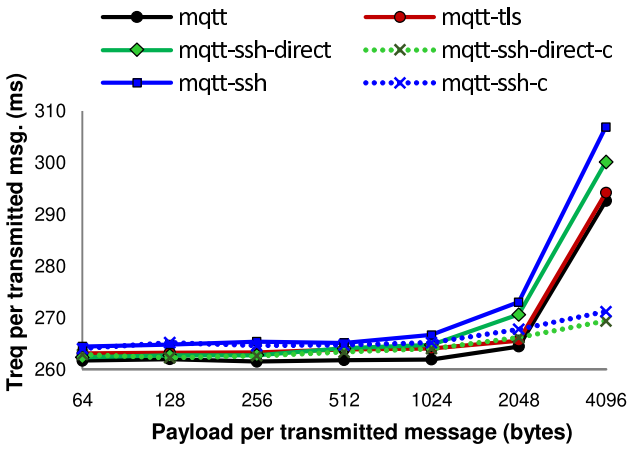

Fig. 7. Treq per transmitted message of each configuration on the lossy channel (125ms delay, 5ms jitter, 1% packet loss) for the corner case. The horizontal axis is in log scale.

only a proxy client (such as SSH) and static traffic rules to provide isolation. Fortunately, the SSH protocol is increasingly being included in these devices, and highly optimized, frequently maintained implementations exist. For instance, WolfSSH [53] is available for multiple constrained hardware (e.g., STM32F2/F4, PIC32, NXP/Freescale), and operating environments (e.g., ThreadX, FreeRTOS, ZephyrOS [64]).

We expect developers and manufacturers to find the CMXsafe proxy layer appealing due to new incoming regulations in some markets. In the United States, the Federal Communication Commission, in line with the National Cybersecurity Strategic Objectives, has proposed a Voluntary Cybersecurity Labeling Program to ensure that the "*IoT devices and products comply with the Commission's program requirements*" [65], [66]. The European Union has gone a step further with the Cyber Resilience Act [67], as some security aspects of IoT devices *must* be certified (in some cases by independent third parties) and have to be re-certified with every update. To alleviate this, CMXsafe can facilitate the incorporation of composite certification schemes (certify a device by certifying each of its elements and the composition procedure) as described by SESIP, a recently approved EU certification standard [68]. This should reduce compliance efforts, as CMXsafe updates can be reused in other IoT devices. This accelerates the patching process, reducing certification costs and preserving the intellectual property of core IoT applications if the procedure is delegated to specialized third parties. It also simplifies the design of core IoT device applications after offloading the security concerns in independent modules, preventing these applications from becoming frequently outdated and increasing the effective lifespan of the devices.

VI. Related Work

The significantly extended threat landscape in IoT environments [5], [8], [69], [70] has motivated the research community to pursue their security concerns from various perspectives. We summarize in the following the closely related to our current focus.

A. SOCKS Protocol

SOCKS protocol [71] is a lightweight multipurpose proxying protocol operating at layer five. A general setup includes the SOCKS protocol client embedded within the application and a remote SOCKS proxy server acting in lieu of the application to establish connections to other systems. Its design, however, causes operational limitations that motivate the CMXsafe proxy layer architecture. In particular, CMXsafe operates at the 4th OSI level and does not require the client application to incorporate another communication protocol or any custom programming in the source code. SOCKS servers, however, only provide proxy functionalities to programs designed in the first place to support the SOCKS protocol, limiting the scope of a SOCKS-based solution. Furthermore, SOCKS does not allow fine-grain the socket allocation on the SOCKS server to limit potential attack surface and lateral movement, as SOCKS provides dynamic allocation ports upon application requests without restriction. Finally, the SOCKS protocol is not devised to provide the functionality of a reverse socket proxy easily. This would require a custom implementation of SOCKS in clients to expose IoT client application services in remote devices or a combination with another protocol that allows reverse socket proxying. Thus, without providing any further technical advantage, this approach would encounter serious limitations in providing the control and isolation level offered by secure communication paths in CMXsafe.

B. VPN/IPsec

Traditionally, VPNs have been established to define secured boundaries or DMZs to isolate unsecured communications between trusted systems through the Internet. This communication scheme allows secure communications with reduced configuration complexity but requires all elements of the VPN to be trusted. VPNs can provide different communication schemes (Host-to-host, host-to-network, or network-to-network) with a host at the lowest granularity level possible. To achieve further granularity levels, other technologies must be combined, increasing the complexity level [72].

CMXsafe operates processing only the specific allowed traffic at a user account/socket level. Instead of being focused on creating boundaries between hosts, CMXsafe allows the establishment of secure communication paths at a socket level (OSI 4). A higher level of granularity and the explicit requisite of a secure communication path to allow communication contribute to reducing the attack surface with enhanced levels of communication isolation. In this regard, VPN isolation has proved insufficient due to vulnerabilities allowing leaked VPN information [73] and attacks leveraging lateral movement. For example, a compromised device belonging to a VPN may be able to access resources in the DMZ and spread the attack to other devices belonging to that VPN [74], [75].

Implementing CMXsafe with VPN technology would result in different challenges, as the target access-control granularity is the socket level. CMXsafe leverages existing features in the network framework to identify the user accounts where proxy processes are run to differentiate traffic. However, VPN

technology uses kernel-managed network interfaces, where no user account source of the communication can be derived. An option to overcome this would include tagging packets before encapsulation at the origin in a similar fashion as MPLS [76], but this would incur further overhead, complexity, and a larger attack surface.

Furthermore, administrations and regulatory bodies are pushing towards communication schemes that do not rely on boundaries or DMZs, like the Zero Trust (ZT) architectures [77]. ZT is a cybersecurity paradigm focused on resource protection and the premise that trust is never granted implicitly and continuously evaluated [78]. *The goal of ZT is to "prevent unauthorized access to data and services coupled with making the access control enforcement as granular as possible"* [79]. Although these objectives can be achieved with VPNs in combination with other technologies [80], [81], the granularity level required suggests a more reasonable approach to this challenge from an upper OSI layer than used by VPNs, as this would reduce complexity. In this line, according to the OBM Memorandum M-22-09 [82], the US government is implementing ZT Architectures detailing specific actions for federal agencies to adopt in alignment with the pillars outlined in the ZT Maturity Model by NIST [79]. Current ZT Reference Architecture promotes a VPN-Less implementation of communication infrastructures as *"In the conventional approach... VPNs pose a threat to enterprise security. After authentication, they create a path in the network perimeter and provide access to network resources"* [83]. In this regard, CMXsafe can contribute to these efforts while also allowing the decoupling of security communications to facilitate security updates.

C. Network-Level Security:

In [84], the authors assume the existence of industrial IoT devices that cannot be updated and suggest customized combinations of countermeasures adapted to each deployment to minimize the cybersecurity risks, including segmentation with DMZs, VPNs, and IPS/IDS mechanisms. In [85], an SDN-based cooperative approach for network attack detection is proposed. In this solution, if an attack is detected by one IoT device, it communicates the acquired knowledge to the logically centralized SDN controller, which will report the attack to the application logic. Then, the SDN controller issues a message to OpenFlow switches to drop the spurious traffic. This proposal aligns with the SDN-enabled logically centralized concept discussed in [86]. Similarly, in [87], the authors proposed an SDN-based architecture where per IoT-device middleboxes are deployed at the data plane level to monitor the generated traffic according to a set of pre-installed security policies reflecting the regular application-layer interactions.

Many efforts have proposed securing IoT environments with gateways deployed at the edge level. In [6], the authors presented an edge security gateway-based architecture empowered with SDN. Based on their adversarial model, the authors defined a set of security properties that need to be satisfied by the architecture to claim its trustworthiness. To guarantee these properties, the authors combine the use of periodic remote attestation to validate the logic and configurations' integrity and micro-hypervisor, critical code isolation, and access control mediation. Barrera et al. [13] leveraged the predictable behavior of consumer IoT devices to derive allowlist network security policies that can be enforced at different points, such as the WAN gateway or the middleboxes sitting between the access points and the gateway. In [14], the authors claim that network security monitoring constitutes a fundamental element to secure IoT environments. In this regard, they presented the DeaBolt architecture, where both low-end and high-end IoT devices are hidden behind access control points.

Those propositions provide dynamic security policy enforcement mechanisms to defend against attack propagation and minimize cybersecurity risks. On the contrary, CMXsafe leverages the predictability of IoT devices' communication to build a secure-by-design communication architecture, where policies are pre-established with fine-grained access control based on predefined permissions. This extends the traffic isolation *within the devices* without requiring modification of the source code of the applications. All major technologies that CMXsafe relies upon are standard and widely implemented in IT/OT systems. This facilitates CMXsafe adoption and its long-term support.

D. Security Decoupling

The risks related to the use of integrated TLS security have been discussed in [11], [88], [89], and [5]. Those works converge to the necessity of simplifying TLS for it to be used securely by applications. To address this, O'Neil et al. proposed a standalone design of TLS API by leveraging the standard POSIX socket APIs of the OS [12].

In a context more specific to IoT environments, the authors in [11] studied the security of TLS usage in IoT devices. They concluded with a set of recommendations to improve the security of IoT devices. In [15], OpenSSL library vendor patching practices have been investigated by examining the library's version in the IoT firmware releases and the OpenSSL update history based on vulnerability discoveries [90]. To address this problem, the authors proposed an architecture supporting the centralized management of third-party libraries.

CMXsafe guarantees full independence to the IoT application source code and its operation. This paves the way for a general patching procedure of security network features of IoT devices thanks to the reusability of CMX-agents. CMXsafe also allows a progressive implementation, as the operation of the proxy layer is transparent to the IoT application and IoT platform. Furthermore, communication between IoT device applications and IoT platforms occurs within CMX-GWs, reducing the attack surface and mutual authentication overhead. Identification of the IoT devices is also included natively in the source IP address of any proxied packet, facilitating the means to IoT platforms for reliable identification of the source of communications.

VII. Conclusion

This paper analyzes the challenges of addressing common security communication problems affecting IoT deployments.

We propose CMXsafe, an IoT secure-by-design communication architecture agnostic to the application protocols. CMXsafe leverages the security decoupling from communications to provide security updates in the communication protocols as well as access control and authentication of communications with services proxied into the CMX-GWs.

To assess the performance of CMXsafe, we presented an implementation based on existing technologies to facilitate its maintenance and integration. We also devised a testbed featuring an MQTT publisher-broker-subscriber paradigm implementing CMXsafe to compare its performance with an alternative integrated security solution based on TLS. The results provided demonstrate a performance boost with different scenarios and message payloads.

In future works, we will explore proxy implementation efforts with QUIC instead of TCP [91]. QUIC will allow CMXsafe-supported deployments to benefit from new security and functional features without the hurdles of the integration of QUIC into each application. This analysis was left for future work motivated by a recent publication of a draft of the SSHv3 protocol [92], whose embodiment will incorporate TLS and QUIC [93]. This implementation leverages QUIC-Go and maps the QUIC streams to SSH channels, providing a qualitative leap in technical features [94]. We consider this contribution relevant as QUIC-Go [95] is being actively developed with considerable community support, which has also been involved in elaborating the documentation and first versions of SSHv3 embodiments. As the changes are substantial when compared to SSHv2, this analysis was left out of the scope of this paper, noting that in any case, CMXsafe implementations would allow replacing portable embodiments of CMXsafe agents using old SSH embodiments with the new ones seamlessly.

We will also investigate potential approaches for orchestrating CMXsafe deployments as microservice meshes. Integrating the proxy layer into Istio [22] or Linkerd [23] would allow CMXsafe to leverage advanced management capabilities offered by these well-established microservice mesh technologies.

REFERENCES

[1] H. Baur et al. (2022). *Ericsson Mobility Report*. Ericcson. [Online]. Available: https://www.ericsson.com/49d3a0/assets/local/reports-papers/mobility-report/documents/2022/ericsson-mobility-report-june-2022.pdf/

[2] W. Zhou et al., "Reviewing IoT security via logic bugs in IoT platforms and systems," *IEEE Internet Things J.*, vol. 8, no. 14, pp. 11621–11639, Jul. 2021.

[3] C. Xenofontos, I. Zografopoulos, C. Konstantinou, A. Jolfaei, M. K. Khan, and K. R. Choo, "Consumer, commercial, and industrial IoT (In)security: Attack taxonomy and case studies," *IEEE Internet Things J.*, vol. 9, no. 1, pp. 199–221, Jan. 2022.

[4] B. Rodrigues, "Luabot: Malware targeting cable modems," Blog, Tech. Rep., 2016. [Online]. Available: https://w00tsec.blogspot.com/2016/09/luabot-malware-targeting-cable-modems.html

[5] O. Alrawi, C. Lever, M. Antonakakis, and F. Monrose, "SoK: Security evaluation of home-based IoT deployments," in *Proc. IEEE Symp. Secur. Privacy (SP)*, San Francisco, CA, USA, May 2019, pp. 1362–1380.

[6] M. McCormack et al., "Towards an architecture for trusted edge IoT security gateways," in *Proc. 3rd USENIX Workshop Hot Topics Edge Comput. (HotEdge)*, 2020, pp. 1–10.

[7] Team82. (2022). *2H State of XIoT Security 2022*. [Online]. Available: https://claroty.com/resources/reports/state-of-xiot-security-2h-2022

[8] C. Kolias, G. Kambourakis, A. Stavrou, and J. Voas, "DDoS in the IoT: Mirai and other botnets," *Computer*, vol. 50, no. 7, pp. 80–84, 2017.

[9] Azure Industrial IoT. (2023). *Azure Industrial IoT Platform*. Microsoft Corp. [Online]. Available: https://github.com/Azure/Industrial-IoT

[10] Amazon AWS. (2020). *AWS IoT Device SDK C. SDK for Connecting To AWS IoT From a Device Using Embedded C. Migration Guide for MQTT*. [Online]. Available: https://aws.github.io/amazon-freertos/202107.00/embedded-csdk/docs/doxygen/output/html/mqtt_migration.html

[11] M. T. Paracha, D. J. Dubois, N. Vallina-Rodriguez, and D. R. Choffnes, "IoTLS: Understanding TLS usage in consumer IoT devices," in *Proc. ACM Internet Meas. Conf. (IMC)*, Nov. 2021, pp. 165–178.

[12] M. O'Neill et al., "The secure socket API: TLS as an operating system service," in *Proc. 27th USENIX Secur. Symp.*, Baltimore, MD, USA, 2018, pp. 799–816.

[13] D Barrera, I Molloy, and H. Huang, "Standardizing IoT network security policy enforcement," in *Proc. Workshop Decentralized IoT Secur. Standards (DISS)*, Reston, VA, USA, 2018, p. 6.

[14] R. Ko and J. Mickens, "DeadBolt: Securing IoT deployments," in *Proc. Appl. Netw. Res. Workshop*, Montreal, QC, Canada, 2018, pp. 50–57.

[15] H. Zhang et al., "Capture: Centralized library management for heterogeneous IoT devices," in *Proc. 30th USENIX Secur. Symp.*, 2021, pp. 4187–4204.

[16] J. Carrillo-Mondéjar, H. Turtiainen, A. Costin, J. L. Martínez, and G. Suarez-Tangil, "HALE-IoT: Hardening legacy Internet of Things devices by retrofitting defensive firmware modifications and implants," *IEEE Internet Things J.*, vol. 10, no. 10, pp. 8371–8394, May 2023.

[17] J. D. De Hoz Diego, J. Saldana, J. Fernández-Navajas, and J. Ruiz-Mas, "IoTsafe, decoupling security from applications for a safer IoT," *IEEE Access*, vol. 7, pp. 29942–29962, 2019.

[18] S. Latvala, M. Sethi, and T. Aura, "Evaluation of out-of-band channels for IoT security," *Social Netw. Comput. Sci.*, vol. 1, no. 1, pp. 63217–63229, Jan. 2020.

[19] M. Azure. (Oct. 2022). *Overview of Azure Hub Device Provisioning Service*. [Online]. Available: https://learn.microsoft.com/en-us/azure/iot-dps/about-iot-dps?view=iotedge-1.4#provisioning-process

[20] F. Kohnhäuser, D. Meier, F. Patzer, and S. Finster, "On the security of IIoT deployments: An investigation of secure provisioning solutions for OPC UA," *IEEE Access*, vol. 9, pp. 99299–99311, 2021.

[21] M. D. Hossain, T. Sultana, G.-W. Lee, and E.-N. Huh, "The role of microservices in the Internet of Things: Applications, challenges, and research opportunities," in *Proc. 14th Int. Conf. Ubiquitous Future Netw. (ICUFN)*, Jul. 2023, pp. 901–906.

[22] ISTIO. (2021). *The ISTIO Service Mesh*. [Online]. Available: https://istio.io/v1.12/about/service-mesh/

[23] Buoyant. (2023). *Linkerd: The World Lightest and Fastest Service Mesh*. [Online]. Available: https://linkerd.io/

[24] J. D. de Hoz Diego et al., "An IoT digital twin for cyber-security defence based on runtime verification," in *Proc. 11th Int. Symp. Leveraging Appl. Formal Methods, Verification Validation Verification Princ. (ISoLA)*. Cham, Switzerland: Springer, 2022, pp. 556–574.

[25] K. L. Ang and J. K. P. Seng, "Application specific Internet of Things (ASIoTs): Taxonomy, applications, use case and future directions," *IEEE Access*, vol. 7, pp. 56577–56590, 2019.

[26] N. Mazhar, R. Salleh, M. Zeeshan, and M. M. Hameed, "Role of device identification and manufacturer usage description in IoT security: A survey," *IEEE Access*, vol. 9, pp. 41757–41786, 2021.

[27] P. Car and S. D. Luca. (2023). *Briefing: EU Cyber-Resilience Act*. [Online]. Available: https://www.europarl.europa.eu/RegData/etudes/BRIE/2022/739259/EPRS_BRI(2022)739259_EN.pdf

[28] Thales Group. (2023). *Why Device Authentication is Necessary for the IoT*. [Online]. Available: https://cpl.thalesgroup.com/faq/internet-things-iot/why-device-authentication-necessary-iot

[29] W. Zhou et al., "Discovering and understanding the security hazards in the interactions between IoT devices, mobile apps, and clouds on smart home platforms," in *Proc. 28th USENIX Secur. Symp.*, 2019, pp. 1133–1150.

[30] Z. Wang et al., "A survey on IoT-enabled home automation systems: Attacks and defenses," *IEEE Commun. Surveys Tuts.*, vol. 24, no. 4, pp. 2292–2328, 4th Quart., 2022.

[31] K. P. Singh et al., "Simplified and secure session key sharing for Internet of Things (IoT) networks," in *Internet of Things and Connected Technologies*. Cham, Switzerland: Springer, 2021, pp. 319–332.

[32] S. S. Panda, D. Jena, B. K. Mohanta, S. Ramasubbareddy, M. Daneshmand, and A. H. Gandomi, "Authentication and key management in distributed IoT using blockchain technology," *IEEE Internet Things J.*, vol. 8, no. 16, pp. 12947–12954, Aug. 2021.

[33] B. Zhao et al., "A large-scale empirical study on the vulnerability of deployed IoT devices," *IEEE Trans. Dependable Secure Comput.*, vol. 19, no. 3, pp. 1826–1840, May 2022.

[34] B. A. Powell, "Role-based lateral movement detection with unsupervised learning," *Intell. Syst. Appl.*, vol. 16, Nov. 2022, Art. no. 200106.

[35] X. Jiang, M. Lora, and S. Chattopadhyay, "An experimental analysis of security vulnerabilities in industrial IoT devices," *ACM Trans. Internet Technol.*, vol. 20, no. 2, pp. 1–24, May 2020.

[36] M. Husnain et al., "Preventing MQTT vulnerabilities using IoT-enabled intrusion detection system," *Sensors*, vol. 22, no. 2, p. 567, Jan. 2022. [Online]. Available: https://www.mdpi.com/1424-8220/22/2/567

[37] M. R. S. Sedghpour and P. Townend, "Service mesh and eBPF-powered microservices: A survey and future directions," in *Proc. IEEE Int. Conf. Service-Oriented Syst. Eng. (SOSE)*, Aug. 2022, pp. 176–184.

[38] B. Yuan, Y. Jia, L. Xing, D. Zhao, X. Wang, and Y. Zhang, "Shattered chain of trust: Understanding security risks in cross-cloud IoT access delegation," in *Proc. 29th USENIX Security Symp. (USENIX Secur.)*, New York, NY, USA, 2020, pp. 1183–1200.

[39] D. Jackson, "Alloy: A language and tool for exploring software designs," *Commun. ACM*, vol. 62, no. 9, pp. 66–76, Aug. 2019.

[40] Amazon Web Services. (2023). *Configuring Mutual TLS Authentication for a REST API—Amazon API Gateway*. [Online]. Available: [Online]. Available: https://docs.aws.amazon.com/apigateway/latest/developerguide/rest-api-mutual-tls.html

[41] M. Brinkmann et al., "ALPACA: Application layer protocol confusion-analyzing and mitigating cracks in TLS authentication," in *Proc. 30th USENIX Secur. Symp.*, 2021, pp. 1–19.

[42] R. Lakshmanan. *Hackers Exploiting Abandoned Boa Web Servers To Target Critical Industries*. Accessed: May 23, 2024. [Online]. Available: https://thehackernews.com/2022/11/hackers-exploiting-abandoned-boa-web.html

[43] OTORIO. (Feb. 2023). *Industrial Wireless IoT—The Direct Path to Your Level 0*. [Online]. Available: https://go.otorio.com/hubfs/Whitepapers and Reports/whitepaper-Industrial wireless IoT research.pdf

[44] Y. Ding, Y. Shi, A. Wang, X. Zheng, Z. Wang, and G. Zhang, "Adaptive chosen-plaintext collision attack on masked AES in edge computing," *IEEE Access*, vol. 7, pp. 63217–63229, 2019.

[45] G. A. Sullivan et al., "Open to a fault: On the passive compromise of TLS keys via transient errors," in *Proc. 31st USENIX Secur. Symp.*, Boston, MA, USA, 2022, pp. 233–250.

[46] M. Dahlmanns, J. Lohmöller, J. Pennekamp, J. Bodenhausen, K. Wehrle, and M. Henze, "Missed opportunities: Measuring the untapped TLS support in the industrial Internet of Things," in *Proc. ACM Asia Conf. Comput. Commun. Secur.*, May 2022, pp. 252–266.

[47] Y. Sheffer, P. Saint-Andre, and T. Fossati, *Recommendations for Secure Use of Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)*, document RFC 9325, Nov. 2022. [Online]. Available: https://www.rfc-editor.org/info/rfc9325

[48] M. Alanazi, A. Mahmood, and M. J. M. Chowdhury, "SCADA vulnerabilities and attacks: A review of the state-of-the-art and open issues," *Comput. Secur.*, vol. 125, Feb. 2023, Art. no. 103028.

[49] L. Song and M. García-Valls, "Improving security of web servers in critical IoT systems through self-monitoring of vulnerabilities," *Sensors*, vol. 22, no. 13, p. 5004, Jul. 2022. [Online]. Available: https://www.mdpi.com/1424-8220/22/13/5004

[50] D. A. Hahn, D. Davidson, and A. G. Bardas, "MisMesh: Security issues and challenges in service meshes," in *Security and Privacy in Communication Networks*. Cham, Switzerland: Springer, 2020. [Online]. Available: https://api.semanticscholar.org/CorpusID:229182499

[51] F. Minna and F. Massacci, "SoK: Run-time security for cloud microservices. Are we there yet?" *Comput. Secur.*, vol. 127, Apr. 2023, Art. no. 103119.

[52] C. M. Lonvick and T. Ylonen, *The SSH Connection Protocol: TCP/IP Port Forwarding*, document RFC 4254, Jan. 2006. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc4254#section-7

[53] WolfSSL. (Dec. 2022). *WolfSSH Lightweight SSH Library*. [Online]. Available: https://www.wolfssl.com/products/wolfssh/

[54] Microsoft. (2023). *Azure RTOS ThreadX Modules Documentation. Chapter 1: Overview*. [Online]. Available: https://learn.microsoft.com/en-us/azure/rtos/threadx-modules/chapter1

[55] Eclipse Foundation. (Oct. 2023). *IoT & Edge Developer Survey Report*. [Online]. Available: https://outreach.eclipse.foundation/iot-edge-developer-survey-2023

[56] OpenBSD Foundation. (Feb. 2023). *OpenSSH Portable Release*. [Online]. Available: https://github.com/openssh/openssh-portable/releases/tag/V_9_2_P1

[57] Eclipse Foundation. (2022). *Mosquitto: An Open-source MQTT Broker*. [Online]. Available: https://mosquitto.org/blog/2022/08/version-2-0-15-released/

[58] P. Thubert et al., *IPv6 Over Low-Power Wireless Personal Area Network (6LoWPAN) Routing Header*, document RFC 8138, Apr. 2017. [Online]. Available: https://www.rfc-editor.org/info/rfc8138

[59] C. M. Lonvick and T. Ylonen, *The SSH Connection Protocol: Channel Mechanisms*, document RFC 4254, Jan. 2006. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc4254#section-5

[60] J. D. de Hoz Diego, J. Saldana, J. Fernández-Navajas, and J. Ruiz-Mas, "Decoupling security from applications in CoAP-based IoT devices," *IEEE Internet Things J.*, vol. 7, no. 1, pp. 467–476, Jan. 2020.

[61] M. Johnston. (Aug. 2022). *Dropbear SSH*. [Online]. Available: https://matt.ucc.asn.au/dropbear/dropbear.html

[62] C. Bormann et al. (Mar. 2024). *Terminology for Constrained-Node Networks*. Internet Engineering Task Force, Internet-Draft draft-bormann-iotops-ietf-lwig-7228bis-00. [Online]. Available: https://datatracker.ietf.org/doc/draft-bormann-iotops-ietf-lwig-7228bis/00/

[63] D. Vlasenko et al. (2023). *BusyBox: The Swiss Army Knife of Embedded Linux*. [Online]. Available: https://www.busybox.net/

[64] WolfSSL. (Feb. 2024). *WolfSSH Adds Support for Zephyr RTOS*. [Online]. Available: https://www.wolfssl.com/wolfssh-adds-support-for-zephyr-rtos/

[65] FCC. (2023). *Cybersecurity labeling for Internet of Things*. Federal Communications Commission. [Online]. Available: https://docs.fcc.gov/public/attachments/FCC-23-65A1.pdf

[66] U.S. DOD. (2023). *National Cybersecurity Strategy*. U.S. Department of Defense. [Online]. Available: https://www.whitehouse.gov/wp-content/uploads/2023/03/National-Cybersecurity-Strategy-2023.pdf

[67] European Commission. (2022). *Regulation of the European Parliament and the Council on Horizontal Cybersecurity Requirements for Products With Digital Elements and Amending Regulation (EU) 2019/1020*. [Online]. Available: https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX:52022PC0454

[68] Eur. Committee for Standardization. (2023). *Security Evaluation Standard for Iot Platforms (SESIP)*. [Online]. Available: https://standards.cencenelec.eu/dyn/www/f?p=205:110:0::FSP_PROJECT,FSP_LANG_ID:74909,25&cs=157BB6AD851BD6F80A208E00FBBD6B8DD

[69] G. Kambourakis, C. Kolias, and A. Stavrou, "The Mirai botnet and the IoT zombie armies," in *Proc. IEEE Military Commun. Conf. (MILCOM)*, Baltimore, MD, USA, Oct. 2017, pp. 267–272.

[70] D. Kumar et al., "All things considered: An analysis of IoT devices on home networks," in *Proc. 28th USENIX Secur. Symp.*, Santa Clara, CA, USA, 2019, pp. 1169–1185.

[71] M. D. Leech, *SOCKS Protocol Version 5*, document RFC 1928, Mar. 1996. [Online]. Available: https://www.rfc-editor.org/info/rfc1928

[72] CloudConnexa. (2024). *OpenVPN 'Enforce Zero Trust Access: Never Trust, Always Verify'*. OpenVPN. [Online]. Available: https://openvpn.net/solutions/use-cases/enforcing-zero-trust/

[73] N. Xue et al., "Bypassing tunnels: Leaking VPN client traffic by abusing routing tables," in *Proc. 32nd USENIX Secur. Symp.*, Anaheim, CA, USA, 2023, pp. 5719–5736.

[74] N. Khantamonthon and K. Chimmanee, "Digital forensic analysis of ransomware attacks on virtual private networks: A case study in factories," in *Proc. 6th Int. Conf. Inf. Technol. (InCIT)*, Nov. 2022, pp. 410–415.

[75] J. Beerman, D. Berent, Z. Falter, and S. Bhunia, "A review of colonial pipeline ransomware attack," in *Proc. IEEE/ACM 23rd Int. Symp. Cluster, Cloud Internet Comput. Workshops (CCGridW)*, May 2023, pp. 8–15.

[76] A. Viswanathan, E. C. Rosen, and R. Callon, *Multiprotocol Label Switching Architecture*, document RFC 3031, Jan. 2001.

[77] J. R. Biden Jr., "Executive order 14028: Improving the nation's cybersecurity," *Daily J. United States Goverment*, vol. 86, nos. 2021–10460, pp. 26633–26647, 2021. [Online]. Available: https://www.federalregister.gov/d/2021-10460

[78] S. M. Scott Rose, O. Borchert, and S. Connelly, "NIST special publication 800-207: Zero trust architecture," Nat. Inst. Standards Technol. (NIST), Gaithersburg, MD, USA, Tech. Rep. 800-207, 2020, doi: 10.6028/NIST.SP.800-207.

[79] Cybersecurity and Infrastructure Security Agency. (2023). *Zero Trust Maturity Model*. [Online]. Available: https://www.cisa.gov/sites/default/files/2023-04/CISA_Zero_Trust_Maturity_Model_Version_2_508c.pdf

[80] J. C. Sapalo Sicato, P. K. Sharma, V. Loia, and J. H. Park, "VPNFilter malware analysis on cyber threat in smart home network," *Appl. Sci.*, vol. 9, no. 13, p. 2763, Jul. 2019. [Online]. Available: https://www.mdpi.com/2076-3417/9/13/2763

[81] Y. Cao, S. R. Pokhrel, Y. Zhu, R. Doss, and G. Li, "Automation and orchestration of zero trust architecture: Potential solutions and challenges," *Mach. Intell. Res.*, vol. 21, no. 2, pp. 294–317, Apr. 2024, doi: 10.1007/s11633-023-1456-2.

[82] Executive Office of the President. (2022). *OBM Memorandum M-22-09: Moving the U.S. Government Toward Zero Trust Cybersecurity Principles*. [Online]. Available: https://www.whitehouse.gov/wp-content/uploads/2022/01/M-22-09.pdf

[83] Defense Information Systems Agency (DISA) and National Security Agency (NSA) Zero Trust Engineering Team. (2023). *DoD Zero Trust Reference Architecture*. [Online]. Available: https://dodcio.defense.gov/Portals/0/Documents/Library/(U)ZT_RA_v2.0(U)_Sep22.pdf

[84] V. Mullet, P. Sondi, and E. Ramat, "A review of cybersecurity guidelines for manufacturing factories in industry 4.0," *IEEE Access*, vol. 9, pp. 23235–23263, 2021.

[85] G. Grigoryan, Y. Liu, L. Njilla, C. Kamhoua, and K. Kwiat, "Enabling cooperative IoT security via software defined networks (SDN)," in *Proc. IEEE Int. Conf. Commun. (ICC)*, Kansas City, MO, USA, May 2018, pp. 1–6.

[86] D. Kreutz, J. Yu, F. M. V. Ramos, and P. Esteves-Verissimo, "ANCHOR: Logically centralized security for software-defined networks," *ACM Trans. Privacy Secur.*, vol. 22, no. 2, pp. 1–36, May 2019.

[87] T. Yu, V. Sekar, S. Seshan, Y. Agarwal, and C. Xu, "Handling a trillion (Unfixable) flaws on a billion devices: Rethinking network security for the Internet-of-Things," in *Proc. 14th ACM Workshop Hot Topics Netw. (HotNets-XIV)*, Philadelphia, PA, USA, 2015, pp. 1–7.

[88] S. Fahl et al., "Rethinking SSL development in an appified world," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, New York, NY, USA, 2013, pp. 49–60.

[89] M. O'Neill et al., "TrustBase: An architecture to repair and strengthen certificate-based authentication," in *Proc. 26th USENIX Security Symp. (USENIX Secur.)*, Vancouver, BC, Canada, 2017, pp. 609–624.

[90] OpenSSL Software Foundation. (2021). *OpenSSL Changelog*. [Online]. Available: https://www.openssl.org/news/changelog.html

[91] D. Bider. (Jun. 2021). *QUIC-based UDP Transport for Secure Shell (SSH)*. [Online]. Available: https://datatracker.ietf.org/doc/draft-bider-ssh-quic/

[92] F. Michel and O. Bonaventure. (Feb. 2024). *Secure Shell Over HTTP/3 Connections*. Internet Eng. Task Force, Internet-Draft draft-michel-ssh3-00. [Online]. Available: https://datatracker.ietf.org/doc/draft-michel-ssh3/00/

[93] F. Michel et al. (Jan. 2024). *SSH3: Faster and Rich Secure Shell Using HTTP/3*. [Online]. Available: https://github.com/francoismichel/ssh3

[94] F. Michel and O. Bonaventure, "Towards SSH3: how HTTP/3 improves secure shells," 2023, *arXiv:2312.08396*.

[95] M. Seemann et al. (Mar. 2024). *A QUIC Implementation in Pure Go*. [Online]. Available: https://github.com/quic-go/quic-go