# Process-Aware Intrusion Detection in MQTT Networks

Philip Empl
philip.empl@ur.de
University of Regensburg
Regensburg, Bavaria, Germany

Fabian Böhm
fabian.boehm@ur.de
University of Regensburg
Regensburg, Bavaria, Germany

Günther Pernul
guenther.pernul@ur.de
University of Regensburg
Regensburg, Bavaria, Germany

## ABSTRACT

Intrusion Detection Systems (IDS) allow for detecting malicious activities in organizational networks and hosts. As the Industrial Internet of Things (Industrial IoT) has gained momentum and attackers become process-aware, it elevates the focus on anomaly-based Network Intrusion Detection Systems (NIDS) in IoT. While previous research has primarily concentrated on fortifying SCADA systems with NIDS, keeping track of the latest advancements in resource-efficient messaging (e.g., MQTT, CoAP, and OPC-UA) is paramount. In our work, we straightforwardly derive IoT processes for NIDS using distributed tracing and process mining. We introduce a pioneering framework called `MISSION` which effectively captures, consolidates, and models MQTT flows, leading to a heightened process awareness in NIDS. Through our prototypical implementation, we demonstrate exceptional performance and high-quality models. Moreover, our experiments provide empirical evidence for rediscovering pre-defined processes and successfully detecting two distinct MQTT attacks in a simulated IoT network.

## CCS CONCEPTS

• **Networks** → *Peer-to-peer protocols*; • **Security and privacy** → **Network security**; **Intrusion detection systems**.

## KEYWORDS

IDS, MQTT, Internet of Things, Distributed Tracing, Process Mining

**ACM Reference Format:**
Philip Empl, Fabian Böhm, and Günther Pernul. 2024. Process-Aware Intrusion Detection in MQTT Networks. In *Proceedings of the Fourteenth ACM Conference on Data and Application Security and Privacy (CODASPY '24), June 19–21, 2024, Porto, Portugal.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3626232.3653271

## 1 INTRODUCTION

Intrusion detection systems (IDS) have long proven themselves as indispensable [68]. They are applicable to many domains [15, 50, 67] and either identify malicious patterns (*signature-based IDS*) or activities deviating from statistically benign behavior (*anomaly-based IDS*) on a host (HIDS) or network (NIDS) [2, 23]. With the proliferation of the Internet of Things (IoT), organizations have increasingly integrated their operational technology (*"physical processes"* [47]) with their IT infrastructure, giving rise to the industrial IoT. Small physical devices and processes shape the IoT but limit performance, communication, and storage capabilities. Keeping the physical processes running is desired to avoid outages [48], necessitating IDS.

Most notably, research initially focused on systems responsible for managing physical processes. Thereby, SCADA (*Supervisory Control and Data Acquisition*) systems mainly constitute an essential part of nowadays's industrial IoT infrastructure [84]. Different efforts on securing parts of industrial SCADA systems like Modbus or DN3P for communication resulted in plenty of anomaly-based, signature-based and specification-based IDS for both hosts [30, 47, 75], and networks [14, 27, 29, 41, 86]. Signature-based IDS cannot detect unknown attacks, and HIDS may infer physical processes as they require agents running on the host machine [47]. This shifts the focus to anomaly-based NIDS for the Industrial IoT. Additionally, as attacks are more advanced and process-aware [51] like Stuxnet [11] or Industroyer (2) [13, 60], physical processes became a baseline for IDS [7, 9, 16, 17, 26, 59].

SCADA systems are well-researched. Still, these systems are nowadays complemented by resource-efficient messaging protocols like MQTT, CoAP, or OPC-UA, allowing the integration of smaller devices throughout the process and shaping a new era of communication [84]. Despite the benefits of these IoT messaging protocols, their adoption has also increased security threats. Almost half of all organizations cannot detect IoT attacks within their networks [28, 44]. These attacks' increasing frequency and sophistication have raised serious security concerns, but IoT messaging protocols and corresponding communication patterns have only been partially explored for NIDS, yet [10, 18, 45, 55].

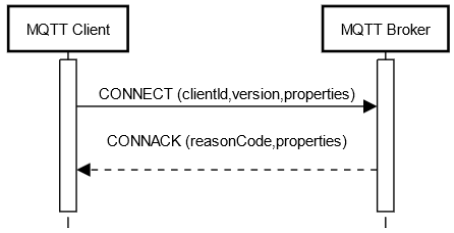
We believe that NIDS can benefit from IoT application layer protocols as they carry more contextual information like topic subscriptions than the transport layer, e.g., TCP. As many MQTT clients (such as sensors and actuators) on a single machine result in multiple ports, attacks are unseen by traditional NIDS. The increasing attack surfaces, the potential for additional contextual information, and the lack of research motivate us to investigate the potential of IoT-specific NIDS. However, using process-aware NIDS with IoT messaging protocols presents two primary challenges:

- First, identifying processes, in general, is an extensive manual task [41], whereby different (structured) data like specifications [8, 41], or network traffic packets [26] allow automation, e.g., creating rules or models. Besides, many IoT devices lead to more packets and network complexity, requiring novel, automatic approaches.
- Second, anomaly-based NIDS are mainly based on artificial intelligence lacking explainability [31], failing because of expertise in training and application within organizations [34], and IoT devices communicate via different, heterogeneous (sub-)networks [1]. Organizations require more traceable, distributed, and easy-to-set-up approaches.

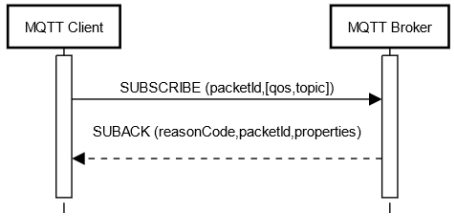


*CODASPY '24, June 19–21, 2024, Porto, Portugal*

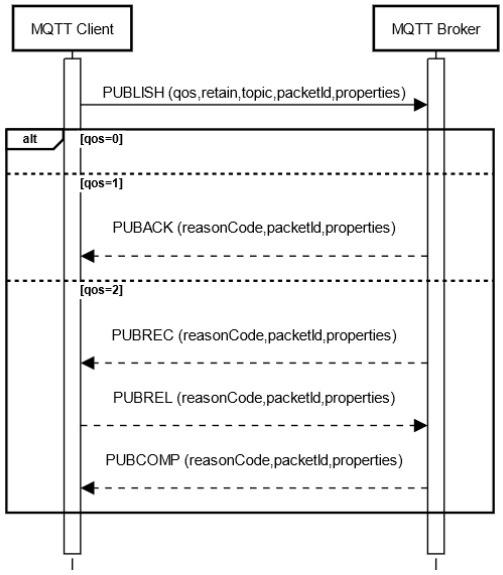
ACM ISBN 979-8-4007-0421-5/24/06.
https://doi.org/10.1145/3626232.3653271




**(a) MQTT CONNECT handshake.**




**(b) MQTT SUBSCRIBE handshake.**




**(c) MQTT PUBLISH handshake.**

**Figure 1: Relevant MQTT handshakes.**

In this paper, we aim to solve these challenges by suggesting network monitoring techniques to automatically and passively record IoT traffic and reduce the number of packets by building flows without risking the physical network's availability. We refer to distributed tracing primarily yet used for auditing [12, 22, 32] and threat investigation and detection [33, 35, 57, 58]. We create explainability and reliability of physical processes as input for NIDS. Consequently, we ask *"How can distributed tracing be utilized to automatically mine IoT processes for NIDS?"* Our main research contributions are the following:

- We envision a distributed and explainable framework `MISSION` to mine IoT processes from network traffic. The process model can be used as an input for NIDS.
- We implement an open-source prototype (GitHub and DockerHub) that allows for real-time probing, collecting, and storing of MQTT flows. Additionally, we provide Jupyter notebooks to showcase the mining of MQTT processes.
- We create an MQTT simulation for research. The simulation consists of multiple sensors and actuators.

This paper is structured as follows. We begin by elaborating on the relevant background for MQTT, IoT network monitoring, and distributed tracing in Section 2. Additionally, related works in this field are discussed in Section 3. In Section 4, we demonstrate the importance of contextual information through two MQTT attack scenarios and the challenges that arise when identifying them. The `MISSION` framework, presented in Section 5, is designed to capture, aggregate, and mine IoT network traffic for NIDS. To evaluate the effectiveness of the `MISSION` framework, we conduct experiments on its performance, process model quality, and its ability to identify two attack scenarios using a prototypical implementation and simulation environment in Sections 6 and 7. Finally, we discuss our research in Section 8 and conclude the paper in Section 9.

# 2 BACKGROUND

## 2.1 Message Queuing Telemetry Transport

MQTT stands for Message Queuing Telemetry Transport. It is a lightweight messaging protocol designed for machine-to-machine or IoT communication. MQTT protocol was invented in 1999 by Andy Stanford-Clark of IBM and Arlen Nipper of Arcom Control Systems [24]. MQTT is a publish/subscribe-based protocol that uses a client/server model. The protocol operates over TCP/IP protocol, allowing devices to communicate over unreliable networks with low bandwidth. MQTT is widely used in IoT applications due to its efficiency, low power consumption, and ability to handle intermittent connections. It is an open standard (ISO/IEC 20922 MQTT 3.1.1 [38] and OASIS MQTT 5.0 [21]) and is used in a wide range of industries, including home automation and industrial automation.

The MQTT 5.0 protocol utilizes a packet structure comprising a fixed header, variable header, and payload. The fixed header includes one of 16 control packet types, such as the CONNECT or CONNACK, and defines relevant flags for each control type. For instance, the PUBLISH packet includes flags like the quality of service (QoS) level and topic. The variable header contains a packet identifier for request-response mapping and properties, such as the `0x08` property defining the response topic and the `0x09` property pertaining to correlation data. The payload carries data that varies depending on the control packet type. In Figure 1, we summarize three key communication handshakes within an MQTT network, with Figures 1a and 1b relying on the connection to an MQTT broker and topic subscription. A request mostly results in an acknowledgment. Figure 1c illustrates the PUBLISH handshake that varies based on the QoS level. A QoS of 0 is equivalent to fire-and-forget, QoS 1 acknowledges the request, ensuring the packet arrives at least once, and QoS 2 ensures that the packet arrives only once.

**Table 1: Exemplary IPFIX fields.**

| ID | Name | Data Type |
|---|---|---|
| 4 | protocolId | unsigned8 |
| 7 | sourceTransportPort | unsigned16 |
| 8 | sourceIPv4Address | ipv4Address |
| 11 | destinationTransportPort | unsigned16 |
| 12 | destinationIPv4Address | ipv4Address |
| 152 | flowStartMilliseconds | dateTime |
| 153 | flowEndMilliseconds | dateTime |
| 161 | flowDurationMilliseconds | unsigned32 |

## 2.2 Network Monitoring

Network monitoring is considered an effective cybersecurity measure [68]. A variety of established standards are available to enable network monitoring. One such standard is the Simple Network Management Protocol (SNMP), outlined in RFC 1157 [39]. SNMP utilizes agent-based probes to gather information, which is subsequently forwarded to central managers for analysis. RFC 1757 [43] describes remote network monitoring management techniques, including proactive monitoring, offline operations, and probes that transmit data to multiple managers. Other available standards include sFlow (RFC 3176 [65]) and Netflow's Version 9 (v9), an open standard developed by Cisco Systems and defined in RFC 3954 [19]. IPFIX, defined in RFC 5153 [3] and RFC 5470 [4], is an extension of Netflow v9 and is commonly referred to as Netflow v10. Developed by the Internet Engineering Task Force (IETF), this standard protocol enables the transfer of flow data from network devices such as routers to a collector for analysis, surpassing the capabilities of its predecessor, NetFlow. Table 1 provides insight into eight of the 491 pre-defined IPFIX fields, which contain information regarding the protocol in use, IP addresses and ports, and flow length. Fields 492-32767 are unassigned, allowing for user-defined ones.

## 2.3 Distributed Tracing

Distributed tracing is a method (see OpenTelemetry [63] or LTTng [72]) used in computer systems to monitor transactions across multiple services. It creates a trace, or complete record, of a request's journey through various microservices or components by assigning a unique identifier, called a trace id, to each incoming request [64, 78]. This id is logged with any relevant metadata by each component and sent to a centralized tracing system. Distributed tracing helps identify bottlenecks, diagnose performance issues, and optimize system resources in large-scale, cloud-native applications with many interconnected services [74].

Distributed tracing is aligned with process mining [49], a popular data-driven approach using event logs to extract insights and knowledge for discovering, analyzing, and improving business processes [80]. Process mining employs data mining, statistics, and visualization techniques to identify inefficiencies, bottlenecks, and compliance issues and suggest ways to optimize them. Information systems typically record event logs. The discovery of processes is facilitated by three primary algorithms, the Alpha miner, Inductive miner, and Heuristic miner, which output Petri nets or heuristic networks that match the input event logs' behavior [81].

**Table 2: IoT network monitoring tools.**

| | Tool | </> | HTTP | MQTT | CoAP | AMQP | XMPP |
|---|---|---|---|---|---|---|---|
| *Flow* | nProbe [62] | • | • | | | | |
| | softflowd [37] | • | | | | | |
| | pmacct [70] | • | | | | | |
| | nfcapd [79] | • | | | | | |
| | Snort [79] | • | | | | | |
| | Zeek [79] | • | • | • | | • | • |
| *Packet* | nDPI [61] | • | • | • | • | • | • |
| | Flowmon [71] | | • | • | • | | |
| | mProxy [53] | • | | • | | | |
| *Subscription* | Telegraf [36] | • | • | • | | • | |
| | Zabbix [85] | • | • | • | | | • |
| | Nagios Core [42] | | • | • | | | • |
| | Nagios XI [42] | | • | • | | | • |
| | Paessler PRTG [66] | | • | • | | | |
| | ManageEngine [54] | | • | | | • | |
| | Site24x7 [76] | | • | | | • | • |
| | SolarWinds [77] | | • | | | • | |

# 3 RELATED WORK

Through process-aware attacks like Stuxnet [11] or Industroyer (2) [13, 60], physical processes became a baseline for IDS [7, 9, 16, 17, 26, 59]. Besides, machine learning approaches for MQTT intrusion detection, e.g., [18, 45], there is to the best of our knowledge currently no intrusion detection mechanisms for IoT messaging protocol claiming reliability and explainability. Casola et al. [10] design a signature-based monitoring of IoT devices' data. Closest to our work, Matoušek et al. [55] define a CoAP IPFIX extension to monitor, statistically analyze, and model IoT flows for NIDS. However, they only take TCP sessions into account and statistically analyze the flow attributes. We go beyond existing research by investigating the MQTT protocol and automatically deriving explainable process models using distributed tracing and process mining for intrusion detection.

Besides, distributed tracing has yet been used for auditing [12, 22, 32] or threat investigation and detection [33, 35, 57, 58]. Especially using process mining for cybersecurity operations is a highly influential research topic [52]. For instance, analyzing network traffic data has already been addressed, e.g., [6]. Wakup et al. [82] showed the transformation of TCP traffic to events logs to mine the protocol's behavior. They referred to this process as protocol mining and did not further abstract the network traffic. From an organizational context, Englberg et al. [25] use process mining in combination with network traffic data to better inform activities in business processes. They suggested a model that aligns network traffic with business processes. Process mining has already been used for analyzing IoT attacks [20]. However, Macák et al. [52] state that real-time processing of network traffic data paired with process mining is in its early stage. To our knowledge, we are the first to use process mining techniques on MQTT network traffic.

Moreover, we analyze various tools for IoT network monitoring on the application layer and categorize them based on their respective capabilities, as presented in Table 2. We identify three types of IoT monitoring techniques, namely subscription-based, packet-inspection, and flow-based tools. While subscription-based tools subscribe to all topics within an IoT network, packet inspection

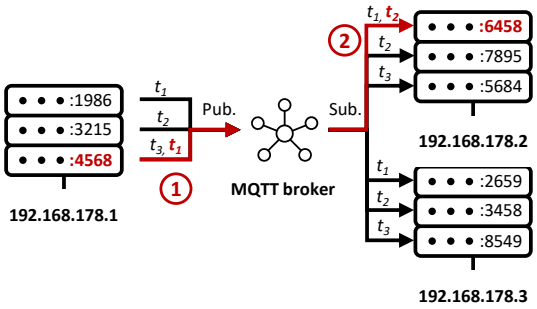



**Figure 2: Threat model considering an MQTT network and two possible scenarios.**

tools perform deep inspections of specific packet payloads. Flow-based tools observe aggregated packets. However, subscription-based and packet-inspection tools have limitations, especially when dealing with large amounts of data generated by IoT devices, as they may lead to significant performance issues. Moreover, messaging brokers used in subscription-based tools may be vulnerable, compromising the data source's authenticity. Flow-based tools for IoT network monitoring address these challenges. They provide a more comprehensive view of the network traffic and can detect anomalies more effectively. Beside Zeek, we are unaware of tools that probe and aggregate MQTT messages using the flow-based approach. To date, Zeek only targets MQTT 3.1.1 and lacks the correlation field, we concentrate on IPFIX-based MQTT 5.0 monitoring.

# 4 ADVERSARY MODEL

## 4.1 Attack Scenarios

This paper considers attacks on IoT networks, especially MQTT. The following discusses the threat model within an MQTT network (see Figure 2). An MQTT network is based on the publish/subscribe architecture of publishers and subscribers. In our proposed network, three devices with different IP addresses run three MQTT clients, with one client publishing data to topics (`192.168.178.1`), while the other two clients (`192.168.178.2` and `192.168.178.3`) subscribe to these topics to receive the data. We define two possible attack scenarios and showcase the associated risks.

The first attack scenario ① involves an attacker gaining unauthorized access to an MQTT client and publishing data to different topics through port `4568`. This may occur due to weak access control or insecure configurations. Attackers can conduct denial-of-service attacks by flooding the MQTT network with MQTT Publish messages. In the second attack scenario ②, malicious MQTT clients subscribe to topics they are not authorized to access through port `6458` on `192.168.178.2`. This attack can happen for various reasons (e.g., misconfigurations in the MQTT broker's access control), allowing an attacker to connect successfully to a MQTT client.

Traditional NIDS based on IP address and port information fail to detect such attacks as the attacker behaves normally and sends messages using a valid MQTT client (IP-port combination). Though access control measures such as constraining topic subscriptions can be enforced, many clients publishing on many topics will result in complexity. In this context, we investigate the efficacy and limitations of network-based graphs to detect these attacks.

## 4.2 Attack Detection

We look closely at an MQTT network's structure and ordinary modeling. We can represent an MQTT network with several communicating devices as a directed graph containing vertices and edges, e.g., by automatically processing a PCAP. The vertices are considered MQTT clients, and the edges as communication links between these clients. We formally define an MQTT network in adoption to Korte et al. [46] as follows:

Theorem 4.1. *An MQTT network structures clients and messages in a directed graph $G = (C, M)$, where*

$$C = \{c_1, c_2, ..., c_n\} \tag{1}$$

*is a finite set of MQTT clients and*

$$M \subseteq \{(x, y) | (x, y) \in C^2, x \neq y\} \tag{2}$$

*defines messages between those clients, containing different attributes $a$, e.g., $a(m_{type})$, $a(m_{topic})$, or $a(m_{properties})$.*

Despite its apparent simplicity, the formal definition of MQTT communication relationships provides a valuable semantic foundation for our research objectives. By leveraging this understanding, we can construct network graphs that reveal fundamental insights into the network's structure. This includes the identification of network centralities, such as brokers, and examining communication patterns between individual devices. As each communication is directed and weighted/labeled, we can deduce the communication's intended purpose. For example, we can discern when a client connects to an MQTT broker and publishes data on different topics.

By utilizing this approach, we can detect attacks occurring in the threat model. For example, if there is no link between 192.168.178.1:4568 and the MQTT broker on topic $t_1$ in scenario ①, we can deduce an anomaly, such as a denial-of-service, as $t_1 \notin a(m_{topic})$, where $m = (192.168.178.1 : 4568,\ MQTT\ broker) \in C$. Similarly, we can identify the attack in scenario ② as there is typically no connection between 192.168.178.1:6458 and the MQTT broker on topic $t_2$: $t_2 \notin a(m_{topic})$, where $m = (MQTT\ broker,\ 192.168.178.1 : 6458) \in C$. We can detect attacks in these scenarios by inferring the MQTT broker's communication through Publish and Subscribe messages.

In summary, we can detect attacks using graphs or comparable representations when the attacker is not process-aware. In these scenarios, attack detection is successful when the attacker's behavior differs from the graph's benign behavior definition. As we have motivated our research on attackers becoming increasingly process-aware, we assume that the attacker is aware of the graph structure. An attacker may flood a topic $t_x \in a(m_{topic})$ where $m$ signifies benign communication between two MQTT clients and $a$ is a benign attribute of $m$, resulting in a denial-of-service attack remaining undetected because there is no deviation from the graph. This motivates our research, as context information is crucial for IoT-specific NIDS. We can concatenate messages or edges in a graph if we know what occurs after client A sends a message. For example, client A sends a message to client B: $t_x \in a_1(m_{topic}) | m(A, B) \in M$. Client B processes the information and informs Client C with

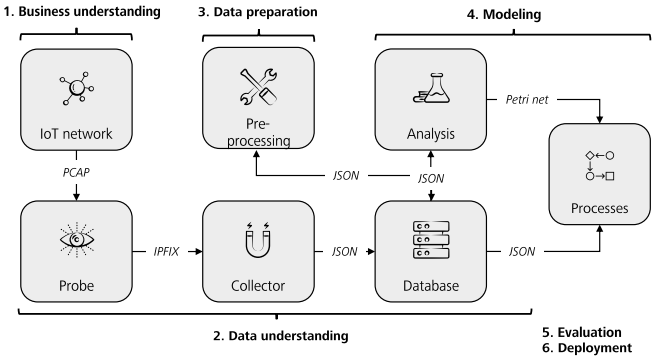



**Figure 3: MISSION framework based on CRIPS-DM [83].**

$t_y \in a_2(m_{topic}) \mid m(B,C) \in M$, resulting in a sequence, or process $P$, where $p_1 = a_1^\frown a_2 \mid p_1 \in P$. In the following, we elaborate on a framework detailing how to deduce such processes.

## 5 MISSION FRAMEWORK

We aim to deduce IoT processes automatically, making them explainable and useful for NIDS without modifying the MQTT 5.0 standard itself. We consider this to be a complex data analysis problem. To ensure rigor, transparency, and traceability, we adopt the *Cross Industry Standard Process for Data Mining* (CRISP-DM) methodology [83], which is highly recognized, aligned with the complexity of the problem and is particularly well-suited for designing data-intensive frameworks in the information systems area [40]. In this paper, we present our approach to mining the processes of IoT networks, especially MQTT, for NIDS, which we call MISSION ("**Mi**ning **S**emantic**s** of **I**oT **N**etworks"). Our framework comprises six phases defined in the CRISP-DM methodology, as illustrated in Figure 3. In the following, we detail the first four phases and provide insights into the phases of evaluation in Section 6.

### 5.1 Business Understanding

First, we aim to understand the problem at hand. Our business problem is the associated cybersecurity risk through process-aware attacks within IoT networks. To target this problem, we mine IoT-specific processes in network traffic. Specifically, we focus on analyzing MQTT networks, elaborating on their processes, and using them for NIDS. We have the following requirements:

- **Explainability.** Models should be explainable by means humans can reproduce their creation and structure.
- **Distribution.** Data should be reliably collected from different (sub-)networks to integrate the whole network.
- **Standards.** Open standards allow for innovation, interoperability, cost-effectiveness, and security.

We address these requirements by employing established distributed tracing methods in combination with process mining and network monitoring standards.

### 5.2 Data Understanding

In the second phase, we need to understand the data. MQTT is a publish/subscribe messaging protocol that operates on the application layer using TCP/IP. In line with this definition, an MQTT network consists of clients ($C$) and messages ($M$) with weighted attributes, such as $a(m_{topic})$. MQTT handshakes are initiated through different packet control types like Connect or Publish. A handshake mostly comprises of a request (e.g., Connect) and an acknowledgment (e.g., ConnAck). We define such handshakes as flows, a "set of related IP packets". In total, MQTT 5.0 has seven possible flows: Connect, Publish, Subscribe, Unsubscribe, Disconnect, Ping, and Authentication. The length of a flow depends on the control packet type. For example, the Publish flow's vary in length depending on the QoS level. A flow with the "at most once" QoS level is made up of one packet, and with the "at least once" QoS level ($QoS = 1$) out of two. In contrast, a flow with the "exactly once" QoS level ($QoS = 2$) comprises four packets. Each packet comprises one or more header fields, additional characteristics, or computationally derived attributes. A flow is identifiable by a key and is unique. Each flow represents an activity within an IoT process instance. We rely on network monitoring techniques to probe, collect, and match related MQTT flows in IPFIX. In the following, we detail *probing*, *collecting*, and *storing* MQTT flows.

**Table 3: Suggested MQTT-specific IPFIX fields.**

| ID | Name | Data Type |
|---|---|---|
| 32769 | mqttQoS | unsigned16 |
| 32770 | mqttControlType | unsigned16 |
| 32771 | mqttPacketId | unsigned16 |
| 32772 | mqttTopic | string |
| 32773 | mqttSrcClientId | string |
| 32774 | mqttDstClientId | string |
| 32775 | mqttCorrelationData | string |

*Probing.* We propose using an IoT network probe to capture and aggregate MQTT network packets, enriching them with contextual information to match related flows and storing them in a database. To transform these packets into flows, MISSION relies on a well-known approach based on 5-tuples, as defined in RFC 6146, which matches packets according to the source IP address, source port, destination IP address, destination port, and protocol:

$$Tuple(srcIP, dstIP, srcPort, dstPort, prot) \tag{3}$$

The combination of IP addresses and ports is considered unique since the MQTT broker maintains the TCP sessions of the MQTT clients, even when multiple clients connect to the broker using the same IP address. Hash tables ("flow tables") match coherent packets and reverse the flow in case of acknowledgments, e.g., inverting source IP and destination IP. As MQTT flows vary in size, the required flow length depends on the packet control type and QoS level, e.g., a Publish flow in QoS level 2 awaits a length of four packets. To prevent collisions resulting from multiple clients' publish requests, the packet identifier complements the 5-tuple if available (only for $QoS > 0$), ensuring that packets are correctly assigned to the corresponding MQTT flow. As MQTT-specific fields are not included in the 491 pre-defined fields of the IPFIX standard, we complement it with user-defined ones (see Table 3). Please note that these fields are suggestions that can be augmented with additional data points. The definition and number of these fields

depends on the users' interest and do not influence our framework. Among these fields, correlation data is of utmost importance as it contains the trace identifier as suggested by distributed tracing. The correlation data should be unique and can be a UUID4 string (an alphanumeric string of 36 characters). MQTT usage has to be modified to transmit the trace identifier. When a client receives correlation data, it must attach it to any potential, following PUBLISH message to keep the trace so that:

$$p = a_1(m_{correlationData}) \frown a_2(m_{correlationData}) \mid p \in P \quad (4)$$

*Collecting & storing.* It is essential to follow a set of steps to ensure the effective and efficient collection and storage of MQTT flows across multiple distributed networks. RFC 5153 provides guidelines for IPFIX collectors, which must be capable of decoding and encoding information, managing templates, and utilizing a transport protocol like UDP or TCP. When probes are widely dispersed across the network, using SCTP as per RFC 4960 is recommended. The IPFIX records should be stored in a document-oriented database that offers more flexibility than traditional relational databases. This flexibility is crucial because flows can vary depending on the application protocol. The data can be stored using JSON serialization to minimize storage requirements and provide easily readable flow records. The collector should use the same template as the probe.

## 5.3 Data Preparation

Once the IPFIX flows are stored in a document-oriented database, preprocessing of the MQTT flows becomes possible. However, two quality aspects need to be considered during preprocessing.

- MQTT flows may not contain client identifiers, especially in PUBLISH or SUBSCRIBE flows.
- MQTT brokers do not have client identifiers, but friendly names must be assigned to devices without one making them human-readable.

In MQTT networks, clients can decide whether a broker retains their sessions, and this information is transmitted via a CONNECT flow. Sessions are managed through specific IP addresses and port combinations within an MQTT broker. Therefore, effective preprocessing must be capable of mapping client identifiers to IP addresses and ports. Additionally, preprocessing must resolve and track MQTT flows with unknown client identifiers. Each MQTT flow defines a source and a destination client. MQTT brokers manage client sessions but do not have client identifiers; hence, they only appear in network traffic data with their IP addresses and ports. To account for this, preprocessing must identify occurrences of MQTT brokers, usually on port 1883 or 8883 (SSL), and assign them a friendly name such as "MQTT Broker". It is important to note that preprocessing may vary if an MQTT broker runs on different ports. Finally, with the collection and preprocessing of MQTT flows completed, the next step is to model and correlate the flows semantically.

## 5.4 Modeling

In the realm of MQTT networks, processes and rules govern their operation. For instance, an industrial process follows business logic and is defined by rules using a control system. These processes may not adhere to strict sequential order and can operate in parallel, resulting in multiple states. Finite-state machines are unsuitable for modeling such networks, as they are designed to model a single client instead of an entire network. On the other hand, Petri nets provide a dynamic model for system behavior and can replicate concurrent processes. These directed networks, comprising vertices and edges, are well-suited to represent MQTT networks. Hence, we adopt the use of Petri nets for modeling MQTT networks, following the model introduced by Petri [69]:

THEOREM 5.1. *MQTT processes structured as a Petri net represent a triple $N = (P, T, A)$, where:*

$$P = \{p_1, p_2, ..., p_i\} \quad (5)$$

*is a finite set of places holding the MQTT clients' state,*

$$T = \{t_1, t_2, ..., t_j\} \quad (6)$$

*is a finite set of transitions representing the PUBLISH flows with attributes like $t_1(m_{topic}, m_{traceId})$, and we can define all arcs of the Petri net as*

$$A = (P \times T) \cup (T \times P) \quad (7)$$

*whereby inputs direct to places and outputs to transitions given weights, so that:*

$$I : t \mapsto p \mid (t, p) \in A \quad (8)$$

$$O : p \mapsto t \mid (p, t) \in A \quad (9)$$

We consider an MQTT client a place so that $c \mapsto p | \, c \in C, p \in P$. We can further specify the MQTT flows gathered in the previous steps as transitions between two MQTT clients, whereby a flow $F$ can be described as $(p, t) \vee (t, p') | (p, t) \vee (t, p') \in A$. A flow may contain different contextual information depending on the packet control type. For instance, PUBLISH flows have fields like $F_{topic,traceId}$, while other packet control types contain different information. Since PUBLISH flows represent the IoT process, the remaining flows, i.e., CONNECT or SUBSCRIBE, are considered prerequisites for publishing messages as there are no PUBLISH flows without prior subscriptions and connections. For instance, due to the publish/subscribe architecture, subscriptions results in PUBLISH flows outgoing from the MQTT broker. Conversely, without authenticating and connecting to the MQTT broker, there are no SUBSCRIBE or PUBLISH flows. By considering only the PUBLISH flows, we can deduce and address the remaining ones.

Besides, the Petri net enables the identification of flows that trigger the traversal of different places, creating new markings. Each MQTT client can concurrently hold multiple markings, describing individual process instances in the Petri net. This is represented as $\forall p : M(p)$, where $M(p)$ denotes the markings held by an MQTT client. The number of markings held by an MQTT client corresponds to the number of concurrent process instances it can handle. The markings enable the tracking of concurrent processes in the MQTT network as they travel through the Petri net. A transition $t$ is activated by the function $F(p, t)$, which involves MQTT clients and subsequent transitions in the Petri net.

Once the formal model has been created, distributed tracing methods, specifically process mining algorithms, can be used to discover Petri nets automatically. These algorithms rely on event logs containing cases or process instances, activities, and additional details such as resources and timestamps [80]. The mapping of IPFIX fields to the event log structure can be represented as follows:

- **Timestamp:** $flowStartNanoSeconds(156)$
- **Case id:** $mqttCorrelationData(32775)$
- **Resource:** $mqttSrcClientId(32773)$
- **Activity:** $mqttTopic(32773)$

Process mining discovery techniques, e.g., Alpha miner, use this event log structure to discover a Petri net of the MQTT network. Additionally, process mining's conformance-checking algorithms - used to compare the behavior of a process as recorded in an event log to the Petri net's behavior - identify deviations acting as NIDS.

## 6 EVALUATION

In this section, we conduct experiments on an implementation based on the MISSION framework to deduce IoT processes for NIDS. Since there are no tools and research on flow-based MQTT 5.0 NIDS, we cannot make a baseline comparison. So, we rely on the attacker scenarios as defined in Section 6.1, and the experimental setting defining IoT processes in Section 6.2. In Section 6.3, we aim to rediscover the pre-defined rules, assess the quality of the Petri nets, and detail how to detect the attacks.

### 6.1 Attack scenarios

Our primary objective is to evaluate the vulnerability of the network to two distinct attack scenarios, as outlined in Section 4. The attacker's objective is to manipulate the network's integrity and confidentiality. The first scenario involves an eavesdropping attack, where an infiltrator compromises network confidentiality by subscribing to MQTT topics assessing the data. The second scenario revolves around an attacker capable of deducing network traffic and executing a Denial of Service (DoS) attack by flooding the network with PUBLISH packets. We aim to identify and analyze both of these attacks with the MISSION framework within an experimental setting, showcasing the power of conformance checking through process mining. The detection of these attacks relies on previously discovered process models. We assume the attacker can access a single node within the network.

### 6.2 Experimental Setting

First, we aim to investigate the applicability of distributed tracing and process mining on flows to derive the IoT processes of an MQTT network. To achieve this, we define a set of rules within an experimental MQTT network and simulate the behavior of several industrial IoT assets derived from a real world use case. We then capture and preprocess the resulting network traffic flows using flow tables (hash tables) within our MQTT probe[1], which exports them to a collector for storage. After preprocessing, we apply the remaining phases of the MISSION framework and its prototypical implementation to unravel the underlying process model.

Our MQTT simulation is implemented using Python and the Eclipse Paho library and is publicly available on GitHub[2]. We simulate an industrial environment consisting of various IoT assets, such as sensors and actuators within different systems like a belt circulation system or a log server, interacting with each other over MQTT. Sensors passively sense the environment and publish data to MQTT topics within a pre-defined value range, while actuators

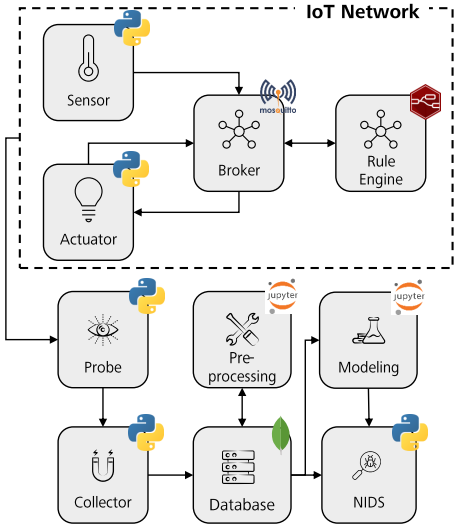


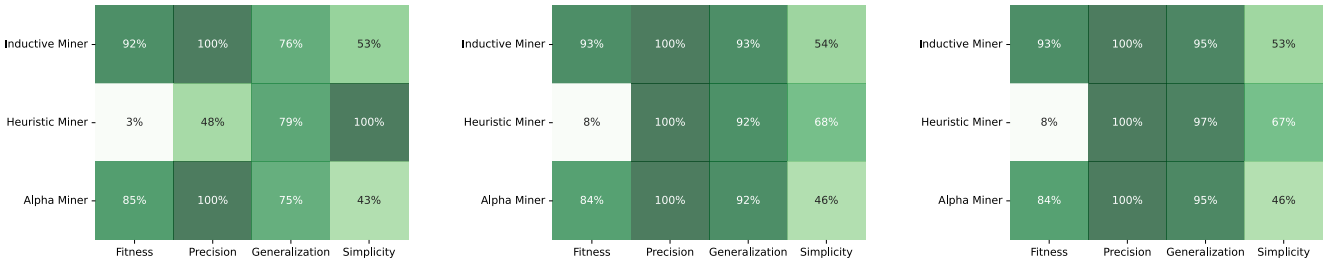

**Figure 4: Experimental setting.**

actively interact with the environment by subscribing to MQTT topics and reacting to incoming events, e.g., state changes from on to off. Status changes are recognized by a rule engine that reacts differently on the base of predefined threshold values. Note that industrial environments typically are static, so we do not consider scenarios where assets join or leave the network. The simulation is deployed on a virtual machine running Ubuntu 22.04 LTS with six cores, 8GB RAM, and 50GB storage. The MQTT probe runs on a separate virtual machine with Ubuntu 20.04.3 LTS (16GB RAM, eight cores, and 80GB storage).

In our MQTT simulation, we instantiate 57 sensors and actuators, including temperature, motion, and window sensors. Each of these devices regularly sends packets at a fixed frequency that depends on the category of the device. The MQTT Broker is implemented using the open-source software Mosquitto, which is widely used in the industry. To connect to the broker, each IoT device in our simulation uses Eclipse Paho, an open-source MQTT client that supports the latest MQTT specification v5 and user-defined fields. To implement the event-driven architecture of our simulation, we use the low-code programming software Node-RED, which acts as an MQTT client and allows the definition of rules. We pre-define three rules in Node-RED within in an industrial context. Note that these rules are notional to exemplify different devices operating.

The probe is deployed within the same network as the MQTT communication, enabling the capture of all MQTT packets. Subsequently, the captured packets are transformed into flows and exported. The collector then aggregates the MQTT flows and stores them in a document-oriented database, MongoDB[3]. Preprocessing of the data is performed after storing the flows. This involves transforming the flows into an event log by mapping IPFIX fields to event log-specific fields, such as case identifiers or timestamps. Additionally, we filtered by PUBLISH flows.

[1]https://github.com/misssion/probe
[2]https://github.com/misssion/simulation
[3]https://www.mongodb.com




**(a) 1,000 events.** **(b) 10,000 events.** **(c) 100,000 events.**

**Figure 5: Evaluation of process mining discovery techniques using different event log sizes.**

## 6.3 Results

The MQTT simulation was conducted over approximately 12 hours, generating a dataset of 100,000 flows. After preprocessing and storing these flows in a MongoDB database, we perform process mining discovery techniques to unravel the IoT processes. We test all three process mining discovery techniques available in the PM4Py[4] library to ensure a thorough comparison of the results. All of our results are available online through a Jupyter notebook[5].

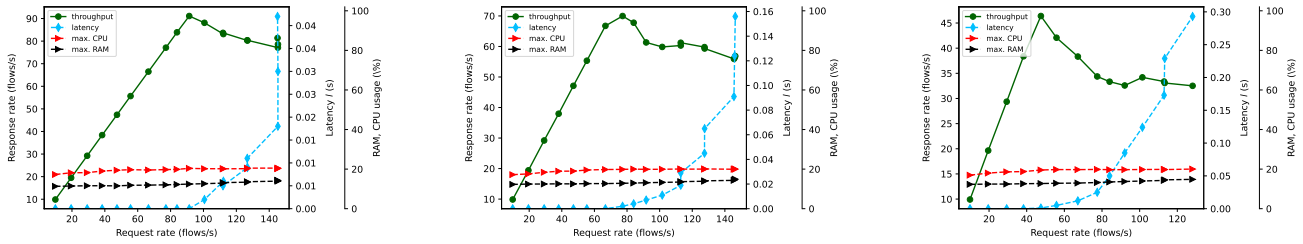
*Directly-follows graph.* Before assessing the quality of the discovered process models in our event log, we discuss using directly-follows graphs (DFG) to generate a model encompassing all available process instances. DFGs capture all process variants and do not aggregate or simplify process models. They can serve as a baseline for comparison with the process mining discovery output or to provide initial insights.

*MQTT processes.* After generating DFGs, we apply process mining discovery techniques to our event log, demonstrating the robustness of these techniques in producing accurate process models. Firstly, we use the Inductive miner on our event log, which excels at identifying relationships within event logs. To exclude MQTT communication that may not be pertinent, we set a noise threshold of 0.9. A high noise threshold only considers continuously occurring and thus probable processes. Secondly, we configure the heuristic miner, which is optimal for working with noisy and incomplete data. The heuristic miner relies highly on a relation threshold, which we set at 0.9. A higher relation threshold eliminates less frequent edges in the process model. Lastly, we employ the Alpha miner, adept at identifying parallel activities. However, selecting the process discovery algorithm highly depends on the amount of distinct events in the network. Within this model, we observe that all pre-defined processes have been successfully re-identified. By measuring and averaging the event log data, we determine that our experimental scenario generates a new process instance every 0.31922 seconds (start time) with a case dispersion ratio of 0.31922 seconds (end time) on average. Our experimental setup can sustainably process the received flows as the arrival and dispersion ratio remains constant. The throughput time is 0.00166 seconds, indicating a near real-time operation.

[4]https://pm4py.fit.fraunhofer.de/
[5]https://github.com/misssion/evaluation

*Model quality.* To assess the quality of the process models discovered by the Inductive miner, Heuristic miner, and Alpha miner, we evaluate them based on four well-established process mining quality criteria: replay fitness, simplicity, precision, and generalization [5]. Figure 5 presents the evaluation results of the three process mining techniques regarding these four criteria. Additionally, we discuss the soundness of our model as an extra evaluation criterion. It is worth noting that we evaluate the models against three different scenarios with varying amounts of flows and process mining discovery techniques. Note that two of our rules occur less frequently. The first rule appears at approximately 4%, and the less frequent rule at approximately 1.2%.

Replay fitness is a well-established quality criterion in process mining, which quantifies the degree of accuracy with which the discovered model can replicate the event log. It measures the percentage of reproduced process instances and ranges from 0% to 100%. A higher percentage indicates better fitness and a value of 100% indicates perfect fitness. Figures 5a, 5b, and 5c demonstrate that the fitness of the discovered process models remains nearly constant across all three process mining techniques. The Inductive and Alpha miners show the best performance regarding fitness, indicating that reproducing the event log would have been feasible with only 1,000 flows. Precision is another essential criterion that measures the overfitting/underfitting of the discovered model. If a model involves more paths than necessary to represent an event log, the model is overfitted. Precision is also measured in percent and reflects the degree to which the model represents the event log. All three models are precise, with at least 10,000 flows. Generalization measures the ability of a discovered model to represent future process instances reasonably. We can see a slight increase in the generalization of the process model with an increase in the number of flows available for process discovery. The simplicity of a discovered model quantifies the maximum complexity required to represent the event log. A higher percentage indicates less complexity needed to represent the event log. We observe that the simplicity of the discovered model stays almost constant although the number of flows increases, except for the Heuristic miner, which shows a substantial decrease in simplicity. Furthermore, we evaluate the soundness of our model using Woflan algorithm, which indicates a binary decision if the discovered model complies with all modeling rules (e.g., start/end activities or no dead locks). In our experimental




(a) MQTT Publish flow with QoS 0. (b) MQTT Publish flow with QoS 1. (c) MQTT Publish flow with QoS 2.

**Figure 6: Benchmarking MQTT probe running on a virtual machine.**

setting, the Inductive miner produces sound models for event logs with at least 10,000 flows. The Heuristic miner has sound models for event logs with more than 100,000 flows. However, the Alpha miner did not produce any sound models.

*Attack detection.* Conformance checking identifies anomalies concerning the event log utilized for process discovery. Initially, we focused on employing procedural conformance checking, e.g., token-based replay (TBR) and alignment checking to detect the two attacks. However, procedural conformance checking does not account for specific resources' involvement in executing activities. Consequently, we encountered challenges in detecting attacks perpetrated by unauthorized resources following a particular process. We attempted declarative conformance checking using a log skeleton as an alternative approach to examine unusual resource-activity allocations within the event log. However, this approach lacked the incorporation of process awareness within declarative models, but also the resource perspective. We decided to rename activities concerning resource-activity allocation to cope with this limitations. Regarding the first attack scenario, the DoS attack, both procedural methods (TBR and alignment) yielded anomalous results, as the attacker did not behave process-aware. This was observed regardless of whether the attacker used the same trace id for packets or unique ones. The log skeleton approach also identified a mismatch between the event log and the malicious traces. In the case of second attack scenario, we successfully detected anomalous subscriptions using procedural methods. However, the declarative method, log skeleton, failed to identify the anomaly when comparing the benign event log with the malicious ones. This discrepancy arose from the activity name "MQTT Broker → mission/log_server-sensor-temperature", which, at first glance, appeared benign due to the publish/subscribe architecture obscuring the client receiving the data. In summary, using process models as a baseline for NIDS requires a well-thought definition of input but heightens the respective process awareness.

## 7 PERFORMANCE EVALUATION

We conduct an assessment of the performance of our MQTT probe. This performance assessment is paramount in determining the probe's ability to process real-time data. To evaluate the probe's performance, we employ the key performance indicators of *maximum sustainable throughput* and *latency*, as proposed by Sedlmeir et al. [73] initially designed for blockchain to measure a node's efficiency. The performance evaluation is documented using Jupyter Notebooks and accessible on GitHub[6] to enable reproducibility.

The maximum sustainable throughput is the highest frequency of network packets that a probe can efficiently transform to IPFIX and export to a collector. To operate the MQTT probe, we deploy a virtual machine running Ubuntu 20.04.3 LTS with 16GB RAM, eight cores, and 80GB disc storage. To determine the maximum throughput, we subject the probe to different fixed frequencies of network packets per second. In a real MQTT network, the probe would receive varying MQTT packets per second, depending on the QoS level. In a network with $QoS = 0$, a probe must export as many flows as the number of packets received. The load increases to two packets in an MQTT network using $QoS = 1$, and four packets are exported for $QoS = 2$. As a result, we create three scenarios where the MQTT probe intends to transform all the packets received. In each scenario, we publish MQTT messages to the MQTT broker at a fixed frequency, beginning with ten packets/s for a minute. The MQTT probe captures the network traffic packets, transforms them into flows, and exports them. We then measure the time $t$ taken to send a message, the export time, the latency ($t_{flow\ exported} - t_{message\ sent}$), and CPU/RAM usage. This process is repeated for different frequencies (step size = 10). Upon successfully executing the experiments, we identify the throughput of each run at a fixed frequency by calculating the linear regressions of responses and requests. The throughput is deemed sustainable and latency-free as long as these linear regressions remain parallel [73]. For instance, throughput is sustainable if the probe receives and exports 40 packets/seconds ($QoS = 0$) with minimum latency. Additionally, we computed the average latency and CPU/RAM usage.

Figure 6 depicts the results obtained from all three scenarios. We transform a total of 92,037 flows for $QoS = 0$, 92,393 flows (184,786 packets) for $QoS = 1$, and 58,675 flows (234,700 packets) for $QoS = 2$. Starting with $QoS = 0$, Figure 6a indicates that our MQTT probe can sustainably export 100 flows/s. After exceeding 100 flows/s, the latency increases, and the response rate decreases. In the $QoS = 1$ scenario (see Figure 6b), the probe can sustainably export 70 flows/s. It is important to note that exporting 70 flows/s with $QoS = 1$ involves a stress factor of 140 packets/s. In the last scenario (see Figure 6c), the maximum sustainable throughput is 50 flows/s (200 packets/s). Interestingly, the RAM and CPU usage remains constant even when the maximum sustainable throughput is achieved. This

[6] https://github.com/misssion/benchmark

phenomenon occurs because our probe is implemented in Python, which has a global interpreter lock to synchronize the execution of threads. Multiprocessing is currently not feasible due to the shared memory of the flow table.

Our MQTT probe performs satisfactorily in all three scenarios, processing between 100 to 200 packets/s and 40 to 100 flows/s, respectively. It is important to note that a collector would aggregate exported flows from multiple probes in a real-world scenario. Scaling to more than one probe would allow for meeting the requirements of more extensive IoT networks. However, in smaller networks, it may be sufficient to operate at least one probe per sub-network to export flows. In summary, our probe, collector and database implementation provides deployable and seamlessly integrated components for existing organizational IoT networks based on MQTT. It focuses on reliability and provides stable and consistent performance for efficient process model discovery and conformance checking on network data enabling security operations.

# 8 DISCUSSION

This section discusses key learnings and insights during our research, limitations in Section 8.1, and future work in Section 8.2.

*OT availability requirements vs. security modifications.* The industrial IoT demands high availability to prevent financial loss resulting from outages [48]. Our framework requires modifications to industrial IoT communication to ensure the trace identifier is transmitted whenever a device reacts to a request. However, this may conflict with the high availability requirements of the industrial IoT. Consequently, organizations must weigh the advantages of improved security against the risk of potential outages, constituting a fundamental decision-making process not limited to MISSION.

*No need for a sledgehammer cracking a nut?* NIDS are shaped by machine learning algorithms that attempt to explain correlations in the data. However, there are cases where it may not be feasible or desirable, particularly in organizations that need more expertise [34] or when models are not explainable [31]. Our more straightforward approach creates reliable and explainable models based on network traffic data and trace identifiers. At present, our framework only generates boolean results for anomalies, whereas machine learning approaches aim to classify attack types directly. Combining both techniques could prove helpful in detecting anomalies and classifying them later, as exemplified by Microsoft Security Copilot's use of large language models [56].

MISSION *as a defensive or offensive measure?* Our research reveals that mining processes in the industrial IoT is feasible with minimal effort, particularly for IoT processes that support business operations. When considering a business process view, manufacturers order numbers to orchestrate their machines, often using plain text communication. These parts of order numbers can also function as trace identifiers resulting in business processes. Our framework, while designed for defensive purposes, could be exploited by attackers to gain insight into operations to launch attacks. It would be interesting to see how attackers have gained process awareness in well-known attacks such as Stuxnet [11] or Industroyer [13, 60]. However, this requires either network traffic in plain text or capabilities for decryption.

## 8.1 Limitations

Although our framework, MISSION, has shown effective in exploring MQTT processes, several limitations should be considered. Firstly, the scope of our paper is limited to MQTT, and other IoT application protocols may have different messaging patterns that require a distinct data collection and modeling approach. Secondly, we have only estimated the flows ($n = 100,000$)needed to derive sound models. Moreover, we only model PUBLISH flows, as they require corresponding SUBSCRIBE or CONNECT flows in advance. The CPU and RAM utilization of the probe and collector has not reached their limits; more personnel resources could enhance their efficiency, for example, through multi-threading or multi-processing. Additionally, we could have integrated our framework into existing probes, such as softflowd or pmacct. However, to maintain the coherence of the framework, we refrained from doing so. Lastly, we consider network traffic in plain text, which often applies to industrial networks. If encrypted, keys must be available.

## 8.2 Future Work

We propose several ideas for further improving and developing the MISSION framework. One such idea is to use business process-relevant information (e.g., product identifiers) to reconstruct business processes instead of using correlation data. Future research should also focus on identifying high-level semantic information to further abstract the process, but also to cope with encrypted traffic. Additionally, research should delve into the alignment of resource-based and control flow-based conformance-checking methods to detect intrusions. While we successfully detected both attacks, the absence of a resource perspective in existing conformance-checking methods poses a challenge, necessitating novel aligned approaches for NIDS. We believe the MISSION framework is getting a cornerstone for IoT security and should be expanded to include additional application protocols such as OPC UA, CoAP, and XMPP. Besides, MISSION can provide proactive security monitoring and testing and identify optimizations for IoT networks, such as mandatory access control rights. For example, MQTT brokers could be optimized based on the process model. Finally, the MISSION framework could be implemented in open-source and commercial probes like softflowd to increase its adoption and get production-ready.

# 9 CONCLUSION

Our paper uses distributed tracing for NIDS to mine IoT processes from network traffic automatically. We found that IoT application protocols contain contextual information relevant to NIDS and can be captured near real-time using network monitoring techniques. We use corresponding flows as input to derive process models using process mining discovery techniques and conformance checking to detect anomalies. Our prototype, including probe, collector, and modeling, is open-source, deployable, and available on GitHub with multi-arch images to be found on DockerHub[7]. We highlight the MISSION framework as a fundamental cornerstone in creating transparency, explainability, and enhancing cybersecurity in IoT networks, enabling more sophisticated security operations beyond network intrusion detection systems.

[7] https://hub.docker.com/u/iotmission

# REFERENCES


[1] Ala I. Al-Fuqaha, Mohsen Guizani, Mehdi Mohammadi, Mohammed Aledhari, and Moussa Ayyash. 2015. Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications. *IEEE Communications Surveys & Tutorials* 17, 4 (2015), 2347–2376. https://doi.org/10.1109/COMST.2015.2444095

[2] Stefan Axelsson. 2000. *Intrusion detection systems: A survey and taxonomy.* Technical Report.

[3] B. Claise and B. Trammell and P. Aitken and S. Zseby and J. Quittek. 2008. *IP Flow Information Export (IPFIX) Implementation Guidelines.* Technical Report. https://doi.org/10.17487/rfc5153 RFC 5153.

[4] B. Trammell and E. Boschi and T. Zseby and D. Quittek and M. Stiemerling and M. Claise. 2009. *Architecture for IP Flow Information Export.* Technical Report. https://doi.org/10.17487/rfc5470 RFC 5470.

[5] Joos C. A. M. Buijs, Boudewijn F. van Dongen, and Wil M. P. van der Aalst. 2012. On the Role of Fitness, Precision, Generalization and Simplicity in Process Discovery. In *Proceedings of the On the Move to Meaningful Internet Systems (OTM 2012)* (2012), Robert Meersman, Hervé Panetto, Tharam Dillon, Stefanie Rinderle-Ma, Peter Dadam, Xiaofang Zhou, Siani Pearson, Alois Ferscha, Sonia Bergamaschi, and Isabel F. Cruz (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 305–322. https://doi.org/10.1007/978-3-642-33606-5_19

[6] Javier Bustos-Jiménez, Cecilia Saint-Pierre, and Alvaro Graves. 2014. Applying Process Mining Techniques to DNS Traces Analysis. In *Proceedings of the 33rd International Conference of the Chilean Computer Science Society, (SCCC 2014).* IEEE Computer Society, 12–16. https://doi.org/10.1109/SCCC.2014.9

[7] Alvaro A. Cárdenas, Saurabh Amin, Zong-Syun Lin, Yu-Lun Huang, Chi-Yen Huang, and Shankar Sastry. 2011. Attacks against process control systems: risk assessment, detection, and response. In *Proceedings of the 6th ACM Symposium on Information, Computer and Communications Security (ASIACCS 2011)* (2011-03), Bruce S. N. Cheung, Lucas Chi Kwong Hui, Ravi S. Sandhu, and Duncan S. Wong (Eds.). ACM, 355–366. https://doi.org/10.1145/1966913.1966959

[8] Marco Caselli, Emmanuele Zambon, Johanna Amann, Robin Sommer, and Frank Kargl. 2016. Specification Mining for Intrusion Detection in Networked Control Systems. In *Proceedings of the 25th USENIX Security Symposium (USENIX Security 2016)*, Thorsten Holz and Stefan Savage (Eds.). USENIX Association, 791–806.

[9] Marco Caselli, Emmanuele Zambon, and Frank Kargl. 2015. Sequence-aware Intrusion Detection in Industrial Control Systems. In *Proceedings of the 1st ACM Workshop on Cyber-Physical System Security (CPSS 2015)*, Jianying Zhou and Douglas Jones (Eds.). ACM, 13–24. https://doi.org/10.1145/2732198.2732200

[10] Valentina Casola, Alessandra De Benedictis, Antonio Riccio, Diego Rivera, Wissam Mallouli, and Edgardo Montes de Oca. 2019. A security monitoring system for internet of things. *Internet of Things* 7 (2019), 100080. https://doi.org/10.1016/j.iot.2019.100080

[11] Thomas M. Chen and Saeed Abu-Nimeh. 2011. Lessons from Stuxnet. *Computer* 44, 4 (2011), 91–93. https://doi.org/10.1109/MC.2011.115

[12] Xutong Chen, Hassaan Irshad, Yan Chen, Ashish Gehani, and Vinod Yegneswaran. 2021. CLARION: Sound and Clear Provenance Tracking for Microservice Deployments. In *Proceedings of the 30th USENIX Security Symposium (USENIX Security 2021)*, Michael Bailey and Rachel Greenstadt (Eds.). USENIX Association, 3989–4006.

[13] Anton Cherepanov. 2017. WIN32/INDUSTROYER: A new threat for industrial control systems. *White paper, ESET (June 2017)* (2017).

[14] Steven Cheung, Bruno Dutertre, Martin Fong, Ulf Lindqvist, Keith Skinner, and Alfonso Valdes. 2007. Using model-based intrusion detection for SCADA networks. In *Proceedings of the SCADA security scientific symposium*, Vol. 46. SRI International, 1–12.

[15] Ronny Chevalier, Maugan Villatel, David Plaquin, and Guillaume Hiet. 2017. Co-processor-based Behavior Monitoring: Application to the Detection of Attacks Against the System Management Mode. In *Proceedings of the 33rd Annual Computer Security Applications Conference (ACSAC 2017)* (2017-12). ACM, 399–411. https://doi.org/10.1145/3134600.3134622

[16] Justyna J. Chromik, Anne Remke, and Boudewijn R. Haverkort. 2016. What's under the hood? Improving SCADA security with process awareness. In *Proceedings of the 2016 Joint Workshop on Cyber- Physical Security and Resilience in Smart Grids (CPSR-SG 2016).* IEEE, 1–6. https://doi.org/10.1109/CPSRSG.2016.7684100

[17] Justyna J Chromik, Anne Remke, and Boudewijn R Haverkort. 2018. Bro in SCADA: Dynamic intrusion detection policies based on a system model. In *Proceedings of the 5th International Symposium for ICS & SCADA Cyber Security Research* (2018-08). BCS Learning & Development, 112–121. https://doi.org/10.14236/ewic/ics2018.13

[18] Ege Ciklabakkal, Ataberk Donmez, Mert Erdemir, Emre Süren, Mert Kaan Yilmaz, and Pelin Angin. 2019. ARTEMIS: An Intrusion Detection System for MQTT Attacks in Internet of Things. In *Proceedings of the 38th Symposium on Reliable Distributed Systems (SRDS 2019)* (2019-10). IEEE, 369–371. https://doi.org/10.1109/SRDS47363.2019.00053

[19] B. Claise, P. Aitken, and N. Ben-Dvora. 2004. *Cisco Systems NetFlow Services Export Version 9.* Technical Report. https://doi.org/10.17487/rfc6759 RFC 3954.

[20] Simone Coltellese, Fabrizio Maria Maggi, Andrea Marrella, Luca Massarelli, and Leonardo Querzoni. 2019. Triage of IoT Attacks Through Process Mining. In *Proceedings of the On the Move to Meaningful Internet Systems (OTM 2019)* (2019) *(Lecture Notes in Computer Science, Vol. 11877)*. Springer, 326–344. https://doi.org/10.1007/978-3-030-33246-4_22

[21] Richard Coppen. 2019. *MQTT Version 5.0 Specification.* Technical Specification. Organization for the Advancement of Structured Information Standards (OASIS). https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html

[22] Pubali Datta, Isaac Polinsky, Muhammad Adil Inam, Adam Bates, and William Enck. 2022. ALASTOR: Reconstructing the Provenance of Serverless Intrusions. In *Proceedings of the 31st USENIX Security Symposium (USENIX Security 2022)* (2023), Kevin R. B. Butler and Kurt Thomas (Eds.). USENIX Association, 2443–2460.

[23] Hervé Debar, Marc Dacier, and Andreas Wespi. 1999. Towards a taxonomy of intrusion-detection systems. *Computer Networks* 31, 8 (1999), 805–822. https://doi.org/10.1016/S1389-1286(98)00017-6

[24] Eclipse Foundation. 2021. Eclipse Newsletter - February 2021. https://www.eclipse.org/community/eclipse_newsletter/2021/february/1.php. Accessed: November 5, 2023.

[25] Gal Engelberg, Moshe Hadad, and Pnina Soffer. 2021. From Network Traffic Data to Business Activities: A Process Mining Driven Conceptualization. In *Proceedings of the 22nd International Conference on Business Process Modeling, Development and Support (BPMDS 2021)* (2021) *(Lecture Notes in Business Information Processing, Vol. 421)*, Adriano Augusto, Asif Gill, Selmin Nurcan, Iris Reinhartz-Berger, Rainer Schmidt, and Jelena Zdravkovic (Eds.). Springer, 3–18. https://doi.org/10.1007/978-3-030-79186-5_1

[26] Robert Flosbach, Justyna Joanna Chromik, and Anne Remke. 2019. Architecture and Prototype Implementation for Process-Aware Intrusion Detection in Electrical Grids. In *Proceedings of the 38th Symposium on Reliable Distributed Systems (SRDS)* (2019-10). IEEE, 42–51. https://doi.org/10.1109/SRDS47363.2019.00015

[27] Igor Nai Fovino, Andrea Carcano, Thibault De Lacheze Murel, Alberto Trombetta, and Marcelo Masera. 2010. Modbus/DNP3 State-Based Intrusion Detection System. In *Proceedings of the 24th IEEE International Conference on Advanced Information Networking and Applications (AINA 2010).* IEEE Computer Society, 729–736. https://doi.org/10.1109/AINA.2010.86

[28] Gemalto. 2019. Gemalto: State of IoT Security. *Network Security* 2019, 2 (2019), 4. https://doi.org/10.1016/S1353-4858(19)30018-2

[29] Niv Goldenberg and Avishai Wool. 2013. Accurate modeling of Modbus/TCP for intrusion detection in SCADA systems. *International Journal of Critical Infrastructure Protection* 6, 2 (2013), 63–75. https://doi.org/10.1016/j.ijcip.2013.05.001

[30] Dina Hadziosmanovic, Damiano Bolzoni, and Pieter H. Hartel. 2012. A log mining approach for process monitoring in SCADA. *International Journal of Information Security* 11, 4 (2012), 231–251. https://doi.org/10.1007/s10207-012-0163-8

[31] Dongqi Han, Zhiliang Wang, Wenqi Chen, Ying Zhong, Su Wang, Han Zhang, Jiahai Yang, Xingang Shi, and Xia Yin. 2021. DeepAID: Interpreting and Improving Deep Learning-based Anomaly Detection in Security Applications. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security (CCS 2021)*, Yongdae Kim, Jong Kim, Giovanni Vigna, and Elaine Shi (Eds.). ACM, 3197–3217. https://doi.org/10.1145/3460120.3484589

[32] Wajih Ul Hassan, Mark Lemay, Nuraini Aguse, Adam Bates, and Thomas Moyer. 2018. Towards Scalable Cluster Auditing through Grammatical Inference over Provenance Graphs. In *Proceedings of the 25th Annual Network and Distributed System Security Symposium, (NDSS 2018)* (2018). The Internet Society. https://doi.org/10.14722/ndss.2018.23141

[33] Wajih Ul Hassan, Mohammad A. Noureddine, Pubali Datta, and Adam Bates. 2020. OmegaLog: High-Fidelity Attack Investigation via Transparent Multi-layer Log Analysis. In *Proceedings of the 27th Annual Network and Distributed System Security Symposium (NDSS 2020)* (2020). The Internet Society. https://doi.org/10.14722/ndss.2020.24270

[34] Khari Hernandez. 2022. For 1 in 4 companies, half of all AI projects fail. https://venturebeat.com/ai/idc-for-1-in-4-companies-half-of-all-ai-projects-fail/

[35] Tim Hübener, Michel R. V. Chaudron, Yaping Luo, Pieter Vallen, Jonck van der Kogel, and Tom Liefheid. 2022. Automatic Anti-Pattern Detection in Microservice Architectures Based on Distributed Tracing. In *Proceedings of the 44th IEEE/ACM International Conference on Software Engineering: Software Engineering in Practice, (ICSE 2022).* IEEE, 75–76. https://doi.org/10.1109/ICSE-SEIP55303.2022.9794000

[36] InfluxData. 2022. Telegraf: MQTT Consumer Input Plugin. https://www.flowmon.com/en/products/software-modules/packet-investigator. Accessed: November 5, 2023.

[37] irino. 2022. softflowd: A flow-based network traffic analyser capable of Cisco NetFlow data export software. https://github.com/irino/softflowd. Accessed: November 5, 2023.

[38] ISO/IEC JTC 1/SC 29/WG 11. 2016. *Information technology – Message Queuing Telemetry Transport (MQTT).* International Standard ISO/IEC 20922:2016. Geneva, Switzerland. https://www.iso.org/obp/ui/#iso:std:iso-iec:20922:ed-1:v1:en

[39] J. Case and K. McCloghrie and M. Rose and S. Waldbusser. 1990. *Simple Network Management Protocol (SNMP).* Technical Report. https://doi.org/10.17487/rfc1448 RFC 1157.

[40] Joyce Jackson. 2002. Data Mining; A Conceptual Overview. *Communications of the Association for Information Systems* 8 (2002), 19. https://doi.org/10.17705/

1cais.00819
[41] Paria Jokar, Hasen Nicanfar, and Victor C. M. Leung. 2011. Specification-based Intrusion Detection for home area networks in smart grids. In *Proceedings of the IEEE Second International Conference on Smart Grid Communications (SmartGridComm 2011)*. IEEE, 208–213. https://doi.org/10.1109/SmartGridComm.2011.6102320
[42] jpmens. 2022. A Nagios/Icinga plugin for testing an MQTT broker. https://github.com/jpmens/check-mqtt. Accessed: 2022-09-27.
[43] K. McCloghrie and M. Rose and S. Waldbusser. 1995. *Remote Network Monitoring Management Information Base*. Technical Report. https://doi.org/10.17487/rfc2819 RFC 1757.
[44] Kaspersky. 2022. *Pushing the limits: How to address specific cybersecurity demands and protect IoT*. Technical Report. Kaspersky.
[45] Muhammad Almas Khan, Muazzam Ali Khan, Sana Ullah Jan, Jawad Ahmad, Sajjad Shaukat Jamal, Awais Aziz Shah, Nikolaos Pitropakis, and William J. Buchanan. 2021. A Deep Learning-Based Intrusion Detection System for MQTT Enabled IoT. *Sensors* 21, 21 (2021), 7016. https://doi.org/10.3390/s21217016
[46] Bernhard Korte and Jens Vygen. 2018. *Graphs*. Springer Berlin Heidelberg, Berlin, Heidelberg, 15–51. https://doi.org/10.1007/978-3-662-56039-6_2
[47] Oualid Koucham, Stéphane Mocanu, Guillaume Hiet, Jean-Marc Thiriet, and Frédéric Majorczyk. 2022. Cross-domain alert correlation methodology for industrial control systems. *Computers & Security* 118 (2022), 102723. https://doi.org/10.1016/j.cose.2022.102723
[48] Tim Krause, Raphael Ernst, Benedikt Klaer, Immanuel Hacker, and Martin Henze. 2021. Cybersecurity in Power Grids: Challenges and Opportunities. *Sensors* 21, 18 (2021), 6225. https://doi.org/10.3390/s21186225
[49] Bowen Li, Xin Peng, Qilin Xiang, Hanzhang Wang, Tao Xie, Jun Sun, and Xuanzhe Liu. 2022. Enjoy your observability: an industrial survey of microservice tracing and analysis. *Empirical Software Engineering* 27, 1 (2022), 25. https://doi.org/10.1007/s10664-021-10063-9
[50] Zhenyuan Li, Qi Alfred Chen, Chunlin Xiong, Yan Chen, Tiantian Zhu, and Hai Yang. 2019. Effective and Light-Weight Deobfuscation and Semantic-Aware Attack Detection for PowerShell Scripts. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security (CCS 2019)*, Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz (Eds.). ACM, 1831–1847. https://doi.org/10.1145/3319535.3363187
[51] Yushan Liu, Xiaokui Shu, Yixin Sun, Jiyong Jang, and Prateek Mittal. 2022. RAPID: Real-Time Alert Investigation with Context-aware Prioritization for Efficient Threat Discovery. In *Proceedings of the 38th Annual Computer Security Applications Conference (ACSAC 2022)*. ACM, 827–840. https://doi.org/10.1145/3564625.3567997
[52] Martin Macák, Lukas Daubner, Mohammadreza Fani Sani, and Barbora Buhnova. 2021. Cybersecurity Analysis via Process Mining: A Systematic Literature Review. In *Proceedings of the 17th International Conference on Advanced Data Mining and Applications (ADMA 2021)* (2022) *(Lecture Notes in Computer Science, Vol. 13087)*, Bohan Li, Lin Yue, Jing Jiang, Weitong Chen, Xue Li, Guodong Long, Fei Fang, and Han Yu (Eds.). Springer, 393–407. https://doi.org/10.1007/978-3-030-95405-5_28
[53] Mainflux. 2022. mProxy is an MQTT proxy. https://github.com/mainflux/mproxy. Accessed: November 5, 2023.
[54] ManageEngine. 2022. RabbitMQ Monitoring. https://www.manageengine.com/products/applications_manager/rabbitmq-monitoring.html. Accessed: November 5, 2023.
[55] Petr Matousek, Ondrej Rysavý, and Matej Grégr. 2019. Security Monitoring of IoT Communication Using Flows. In *Proceedings of the 6th Conference on the Engineering of Computer Based Systems (ECBS 2019)*, Maria-Iuliana Dascalu, Ondrej Rysavý, Constanta-Nicoleta Bodea, Moshe Goldstein, and Miodrag Dukic (Eds.). ACM, 18:1–18:9. https://doi.org/10.1145/3352700.3352718
[56] Microsoft. 2023. Introducing Microsoft Security Copilot: Empowering defenders at the speed of AI. https://blogs.microsoft.com/blog/2023/03/28/introducing-microsoft-security-copilot-empowering-defenders-at-the-speed-of-ai/. Accessed: November 5, 2023.
[57] Francesco Minna, Agathe Blaise, Filippo Rebecchi, Balakrishnan Chandrasekaran, and Fabio Massacci. 2021. Understanding the Security Implications of Kubernetes Networking. *IEEE Security & Privacy* 19, 5 (2021), 46–56. https://doi.org/10.1109/MSEC.2021.3094726
[58] Sasho Nedelkoski, Jorge Cardoso, and Odej Kao. 2019. Anomaly Detection and Classification using Distributed Tracing and Deep Learning. In *Proceedings of the 19th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, (CCGRID 2019)*. IEEE, 241–250. https://doi.org/10.1109/CCGRID.2019.00038
[59] Jeyasingam Nivethan and Mauricio Papa. 2016. A SCADA Intrusion Detection Framework that Incorporates Process Semantics. In *Proceedings of the 11th Annual Cyber and Information Security Research Conference (CISRC 2016)*, Joseph P. Trien, Stacy J. Prowell, John R. Goodall, and Robert A. Bridges (Eds.). ACM, 6:1–6:5. https://doi.org/10.1145/2897795.2897814
[60] Nozomi Networks. 2022. *OT/IT Security Report: Cyber War Insights, Threats and Trends, Recommendations*. Technical Report. Nozomi Networks.
[61] ntop. 2022. nDPI: Open and Extensible LGPLv3 Deep Packet Inspection Library. https://www.ntop.org/products/deep-packet-inspection/ndpi/.
[62] nTop. 2022. nProbe - An Extensible NetFlow v5/v9/IPFIX Probe for IPv4/v6. https://www.ntop.org/products/netflow/nprobe/. Accessed: November 5, 2023.
[63] OpenTelemetry. 2023. High-quality, ubiquitous, and portable telemetry to enable effective observability. https://opentelemetry.io/. Accessed: November 5, 2023.
[64] OpenZipkin. 2023. Zipkin. https://zipkin.io/. Accessed: November 5, 2023.
[65] P. Phaal and S. Panchen and N. McKee. 2001. *InMon Corporation's sFlow: A Method for Monitoring Traffic in Switched and Routed Networks*. Technical Report. https://doi.org/10.17487/rfc3176 RFC 3176.
[66] Paessler AG. 2022. PRTG Manual: MQTT Subscribe Custom Sensor. https://www.paessler.com/manuals/prtg/mqtt_subscribe_custom_sensor. Accessed: November 5, 2023.
[67] Aditya Pakki and Kangjie Lu. 2020. Exaggerated Error Handling Hurts! An In-Depth Study and Context-Aware Detection. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security (CCS 2020)*, Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna (Eds.). ACM, 1203–1218. https://doi.org/10.1145/3372297.3417256
[68] Palo Alto Networks. 2022. *The Connected Enterprise: IoT Security Report 2021*. Technical Report. Palo Alto Networks.
[69] Carl Adam Petri. 1966. Communication with automata. https://doi.org/10.21236/ad0630125
[70] pmacct. 2022. pmacct is a small set of multi-purpose passive network monitoring tools. https://github.com/pmacct/pmacct. Accessed: November 5, 2023.
[71] Progress Flowmon. 2022. Flowmon Packet Investigator: Automated PCAP capture and analyzer. https://www.flowmon.com/en/products/software-modules/packet-investigator. Accessed: November 5, 2023.
[72] LTTng Project. 2023. LTTng: Linux Trace Toolkit Next Generation. https://lttng.org/. Accessed: November 5, 2023.
[73] Johannes Sedlmeir, Philipp Ross, André Luckow, Jannik Lockl, Daniel Miehle, and Gilbert Fridgen. 2021. The DLPS: A New Framework for Benchmarking Blockchains. In *Proceedings of the 54th Hawaii International Conference on System Sciences (HICSS 2021)* (2021). ScholarSpace, 1–10. https://doi.org/10.24251/hicss.2021.822
[74] Benjamin H. Sigelman, Luiz André Barroso, Mike Burrows, Pat Stephenson, Manoj Plakal, Donald Beaver, Saul Jaspan, and Chandan Shanbhag. 2010. *Dapper, a Large-Scale Distributed Systems Tracing Infrastructure*. Technical Report. Google, Inc. https://research.google.com/archive/papers/dapper-2010-1.pdf
[75] Amit Kumar Sikder, Hidayet Aksu, and A. Selcuk Uluagac. 2017. 6thSense: A Context-aware Sensor-based Attack Detector for Smart Devices. In *Proceedings of the 26th USENIX Security Symposium (USENIX Security 2017)*, Engin Kirda and Thomas Ristenpart (Eds.). USENIX Association, 397–414.
[76] Site24x7. 2022. RabbitMQ Monitoring. https://www.site24x7.com/plugins/rabbitmq-monitoring.html. Accessed: November 5, 2023.
[77] SolwarWinds. 2022. RabbitMQ Monitoring Tool. https://www.solarwinds.com/server-application-monitor/use-cases/rabbitmq-monitoring. Accessed: November 5, 2023.
[78] Inc. Uber Technologies. 2023. Jaeger. https://www.jaegertracing.io/. Accessed: November 5, 2023.
[79] Ubuntu Manpage Repository. 2022. nfcapd - netflow capture daemon. https://manpages.ubuntu.com/manpages/bionic/man1/nfcapd.1.html. Accessed: November 5, 2023.
[80] Wil Van Der Aalst. 2012. Process Mining. *Commun. ACM* 55, 8 (2012), 76–83. https://doi.org/10.1145/2240236.2240257
[81] Wil Van Der Aalst. 2016. *Process Mining - Data Science in Action*. Springer. https://doi.org/10.1007/978-3-662-49851-4
[82] Christian Wakup and Jörg Desel. 2014. Analyzing a TCP/IP-Protocol with Process Mining Techniques. In *Proceedings of the 2014 International Conference on Business Process Management (BPM 2014)* (2015) *(Lecture Notes in Business Information Processing, Vol. 202)*, Fabiana Fournier and Jan Mendling (Eds.). Springer, 353–364. https://doi.org/10.1007/978-3-319-15895-2_30
[83] Rüdiger Wirth and Jochen Hipp. 2000. CRISP-DM: Towards a standard process model for data mining. In *Proceedings of the 4th international conference on the practical applications of knowledge discovery and data mining*, Vol. 1. Manchester, 29–40.
[84] IIoT World. 2022. 2022 Building IIoT Systems Survey Report. https://www.iiot-world.com/wp-content/uploads/2022/10/2022-Building-IIoT-Systems-Survey-Report.pdf Accessed: November 5, 2023.
[85] Zabbix. 2022. Zabbix + MQTT. https://www.zabbix.com/de/integrations/mqtt. Accessed: November 5, 2023.
[86] Chunjie Zhou, Shuang Huang, Naixue Xiong, Shuang-Hua Yang, Huiyun Li, Yuanqing Qin, and Xuan Li. 2015. Design and Analysis of Multimodel-Based Anomaly Intrusion Detection Systems in Industrial Process Automation. *IEEE Transactions on Systems, Man, and Cybernetics: Systems* 45, 10 (2015), 1345–1360. https://doi.org/10.1109/TSMC.2015.2415763