Checking Information Flow in Cloud-based IoT Access Control Policies

***Abstract*—Many cloud providers for IoT technologies offer access control mechanisms whose proper configuration is critical for security. However, verifying permissions in isolation is insufficient in a setting where devices have different levels of trust or are compartmentalized in various subsystems. This work analyses IoT access control policies to identify and mitigate potential security vulnerabilities from unwanted information flow between devices. To this end, we present a formal model of AWS IoT Core's components and show how to construct an information flow graph to capture communication interactions from device access control policies, thus enabling the verification of information flow between devices. We implement our approach in a tool called IoT:Poker, and assess it on several real-world IoT access policies.**

***Index Terms*—Information-flow Controls, Policy Verification, IoT-Cloud Security**

I. Introduction

Cloud computing has emerged as a cornerstone for Internet of Things (IoT) deployments, offering scalable infrastructure and platform services tailored to IoT needs: numerous cloud providers offer specialized Platform-as-a-Service and Infrastructure-as-a-Service solutions designed for IoT applications, e.g., AWS IoT Core [1] and Azure IoT Hub [2]. These services allow IoT developers to offload security responsibilities and deployment complexities onto cloud providers.

Regarding security, many providers supply developers with languages to define their access control policies (*IoT policies*). An IoT policy is a specification of what resources an IoT device can access, which actions it can perform, and under which conditions (e.g., publish an MQTT [3] message – a type of action from a popular IoT messaging protocol, see Section II). Properly configured policies are essential for the security of IoT systems. Indeed, previous research has highlighted the susceptibility of IoT policies to misconfigurations and the significant risks they may cause [4]. However, preventing unauthorized accesses is necessary but not sufficient to ensure security in a setting where devices have a different degree of trust and of robustness against attacks. Therefore, developers resort to compartmentalization to increase the security level of their systems. This requires isolating trusted devices from untrusted ones by removing unwanted dependencies. We say there is an information flow between devices A and B if A can produce a message on which B depends even indirectly.

This paper addresses the problem of checking information flow in IoT access policies, namely, protecting critical devices from untrusted ones and, thereby, enforcing security properties, including confidentiality and integrity. We provide a formal model that characterizes how access to resources in an IoT-Cloud deployment is regulated via policies. Then, we introduce a novel verification procedure to check that a policy enjoys such information flow properties and implement it in a tool called IoT:Poker. Formal verification techniques have been proven effective in detecting misconfigurations and verifying security properties in access control policies at various levels. Indeed, prior works [4]–[8] studied the problem of permission misconfiguration in both cloud-based and IoT access policies, focusing on AWS IAM and AWS IoT Core. However, such works analyze policies in isolation to check the granted permissions and do not consider possible interactions among devices, so possible information flow (see Section VII). Thus, prior policy analysis tools could not effectively analyze functional and security information flow properties. Here, we fill this gap providing a triple contribution: (i) We introduce a formal model of IoT policies, in particular targeting AWS IAM policies and their evaluation in the AWS IoT Core infrastructure. This model is the basis of our verification procedure: it helps us highlight some counterintuitive behaviour arising from the usage of wildcard characters that are resolved at different times during the evaluation of a request. (ii) Given a set of IoT policies, we analyze them and build an *information flow graph* to capture the possible communication interaction between devices allowed by the policy. Checking an information flow is thus reduced to reachability over this graph. We introduce a symbolic version of the graph to make verification feasible in practice. (iii) Finally, we implement our verification mechanism in the tool IoT:Poker, available online [9]. We evaluate its effectiveness and performance by experimenting with it on real-world IoT policies available online [10].

Plan of the paper: In Section II, we introduce the MQTT protocol and AWS IoT Core. Section III exemplifies our approach through a case study. Section IV formalizes AWS IoT Core's main components. In Section V, we describe how to build the information flow graph from a set of policies. Section VI presents our tool and its experimental evaluation. In Sections VII, VIII and IX, we compare our approach with the literature, discuss its possible extensions and limitations and draw some conclusions. Appendixes contain the proofs of our formal development and some complementary results of experimental evaluation of IoT:Poker.

II. Background

MQTT Protocol: Message Queuing Telemetry Transport (MQTT) [3] is a lightweight client/server protocol relying

on a publish/subscribe communication model, designed for environments where resources and bandwidth are constrained and limited, such as machine-to-machine (M2M) and IoT contexts. In the publish/subscribe communication model, the entities sending messages (*the publishers*) and the entities receiving them (*the subscribers*) indirectly interact through the infrastructure provided by a third component (*the broker*), which is the only one responsible for delivering messages. This model provides space, time, and synchronization decoupling: clients do not need to know each other, and any communication between them is asynchronously interceded by the broker, which can store messages for temporarily unavailable clients.

In more detail, the protocol prescribes that a client sends a `Con` request to establish a connection with the broker. The request contains a client id, which identifies the client to the broker. (Of course, a client can disconnect by sending a `Disc` message.) After connection, clients communicate by publishing and subscribing messages over *topics:* virtual communication channels managed by the broker. In practice, topics are hierarchically-structured strings, organized into *topic levels* through forward-slash characters '/'. For example, the topic `floor1/room1/temperature` can be used to communicate the sampled temperature in the given room and floor of a building. The broker performs a pattern-matching activity to filter and forward incoming messages to interested clients. A client can subscribe to a given set of topics by sending a `Sub` request to the broker, specifying a *topic filter*, namely a restricted regular expression denoting the topics it is interested in. (A client can unsubscribe by sending a `UnSub` request.) Clients may subscribe to multiple topics at once, using either the single-level '+' or the multi-level '#' wildcard characters in the string specifying the topic filter: '+' replaces any single topic level, whereas '#' represents any sequence of topic levels. For example, the topic filter `floor1/+/temperature` allows a client to receive temperature values from any room on the first floor of a building. For publishing on a given topic, clients send a `Pub` message specifying the topic and a payload; the broker then propagates the payload to all clients subscribed to that topic. For example, a temperature sensor in `room1` can use the topic `floor1/room1/temperature` to inform all the clients interested in that temperature value.

AWS IoT Policies: Amazon Web Services (AWS) [11] provides cloud services and a communication infrastructure on which IoT developers can deploy their devices, known as AWS IoT Core [1]. Among the various technologies provided by AWS IoT Core, we find the MQTT protocol: the cloud acts as a broker between IoT devices, handling connections, and delivering messages, while providing security mechanisms via the implementation of authentication and access control. Devices may connect to the broker only if they successfully authenticate through a certificate or via other Amazon authentication services [12], [13]; moreover, even when a client is authenticated, access to the MQTT infrastructure is controlled via a collection of access control policies, which selectively restricts the available topics that any given client may use.

IoT Core allows the definition of many different access rights, which correspond to MQTT actions. Here, we only consider the main ones: (i) `iot:Connect` to connect to the cloud infrastructure; (ii) `iot:Publish` to publish on a certain topic; (iii) `iot:Subscribe` to subscribe to some topic filter; (iv) `iot:Receive` to receive the messages published on the specified topics. Note that clients need both the `iot:Receive` and `iot:Subscribe` access rights to receive messages on a given topic. The access rights are specified in a policy (a JSON document) consisting of a set of *policy statements* (access control rules). Each statement describes whether a specific action, e.g. `iot:Publish`, is permitted or not (`Allow` or `Deny`) on a given topic (called resource). Users can use wildcard characters '?' and '$*$' in their policies to represent any single character and any sequence of characters, respectively. Moreover, they may use variables, such as `${iot:ClientId}` to denote attributes associated with the certificate provided by the connecting client. For example, a policy composed of the following statements

```
{ "Effect": "Allow",
  "Action": [ "iot:Subscribe", "iot:Receive" ],
  "Resource": [ "floor1/*/temperature", "floor1/room1/*" ]}
```

grants a client permission to subscribe and receive messages on topics concerning the temperature of any room on the first floor and any sensor in `room1` of the first floor. When a client sends a MQTT connection message, the IoT Core policy evaluation engine instantiates the policy, replacing the variables associated with the certificate provided by the connecting client to obtain a policy with no variables on which permissions are evaluated. The policy evaluation follows a *default-deny* strategy: access is granted if there is at least a policy statement allowing the requested action and no statement denying it; in any other case, access is rejected. For example, suppose that the policy above is augmented with a statement whose effect is to reject any subscription to the topic filter `floor1/+/temperature`. In that case, this last statement takes priority: the device can still subscribe to `floor1/room1/temperature` but not to `floor1/+/temperature`. We refer the reader to the documentation for further details that are immaterial to our formal development [1].

III. IoT:Poker at work on a Smart Building

Assume a smart building of two floors where several subsystems, such as energy consumption, physical access control, heating, and fire alert, are all integrated. Our Building Automation System (BAS) is cloud-based, and it is organised in two layers (see Figure 1): the field layer contains off-the-shelf IoT devices that interact with the physical world via sensors and actuators, while the control layer consists of an MQTT broker and several cloud services that implement the communication infrastructure and the control logic. These services execute appropriate actions when triggered by events occurring in the field layer and provide operators with means to monitor, configure, and control the whole system via dashboards. Moreover, we group IoT devices in subsystems according to their functionalities. For example, smoke detectors are part of

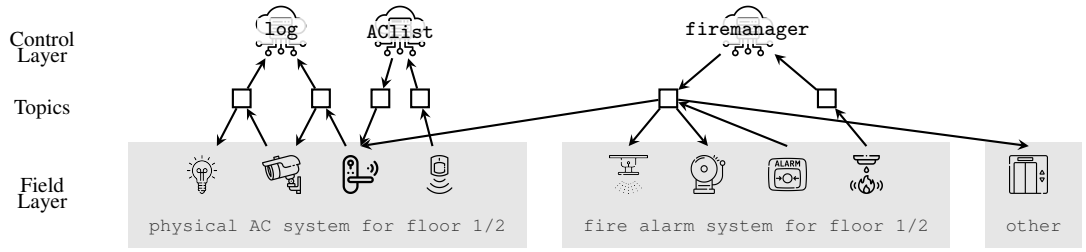

Figure 1. The Architecture of Our Building Automation System.

the fire alarm system, whereas badge readers are part of the physical access control system.

The physical access control system includes a badge reader, a smart lock, a presence sensor, and a light bulb; its expected behaviour for the first floor follows. When the badge reader scans a bar code, it publishes a message on the topic `phAC/floor1/bdgReader1/check`. The cloud service `AClist` waits for messages on this topic and checks the code's validity in the message's payload. If the badge is authorised to open the door, `AClist` publishes a message on the topic `phAC/floor1/lock1/open`. When the smart lock `lock1` receives such a message, it unlocks itself and activates the presence sensor of the room by publishing on the topic `phAC/floor1/prsSens1/enable`. When the presence sensor `prsSens1` is active and detects someone inside the room, it publishes on the topic `phAC/floor1/dtdMovement/light1`. The light bulb `light1` is subscribed to this topic and switches itself on upon reception. The service `log` subscribes to all the topics mentioned above to keep a trace of such events.

The fire alarm subsystem consists of a smoke sensor, an alarm button, an acoustic alarm, and a water pump. Its expected behaviour for the first floor follows. When a smoke sensor detects some smoke in its area, it publishes a message on the topic `fire/floor1/smokeLvl` that is received by the cloud service `fireMngr` subscribed on the related topic filter. If the level of detected smoke exceeds a given threshold, `fireMngr` publishes on the topic `fire/detected` to raise the fire alarm. The water pumps, the acoustic alarms, the doors, and the elevators all subscribe to this topic. When they receive a message on it, the water pumps and the acoustic alarms activate themselves, doors unlock themselves, and the elevator reaches the ground floor, opens the door, and halts. The alarm can also be raised by the fire alarm button that publishes on the same topic.

We assume the BAS system to be configured with a known IoT policy for each device [9]. We aim to assess the correctness of these policies according to two properties: (i) the expected behaviour of the devices must be allowed by their actual access control configuration (to allow them to perform their tasks); (ii) information flow must be forbidden when unnecessary, especially if it may damage the security of the building. We consider a threat model where the cloud infrastructure is trusted, but devices are not, since attackers can compromise them. Moreover, we do not expect access

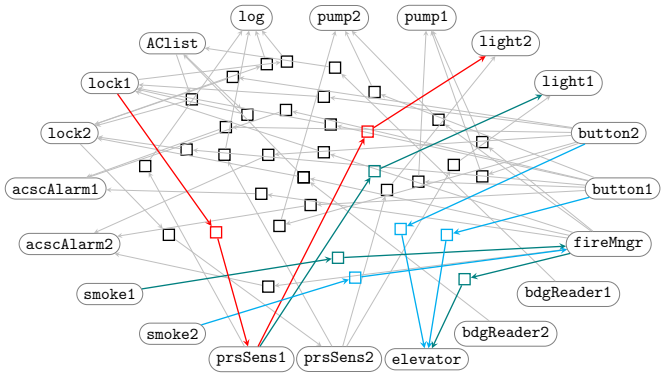

Figure 2. Information Flow Graph for the Building Automation System.

to the code, rather we assume that devices try to publish on and subscribe to any possible topic. Thus, our analysis relies only on the cloud access control policies to verify our target properties. However, analysing policies in isolation and checking permissions are not enough to prove our properties, but we need to take care of all device interactions and consider the policies altogether. Our tool IoT:Poker analyses the devices' policies, reconstructs the permitted device interactions, representing them into an information flow graph, and allows developers to ask queries on possible communications. The graph nodes represent devices and topics, while arcs represent the fact that a device can publish to/receive from a given topic. Figure 2 shows the graph of our BAS, where rounded nodes represent the devices identified by their certificates, while squares represent (symbolic versions of) MQTT topics.

Table I shows some queries along with the output of IoT:Poker. The first two `reach` queries ask the tool to certify that the devices that must communicate are permitted to do so. For example, the query in the first row checks that the sensor `prsSens1` can make the light bulb `light1` switch the light on when someone is in the room; while the second one verifies that the fire alarm can affect the behaviour of the elevator, which is expected to reach the ground floor and halt. Both these queries are satisfied, as the devices can interact, as reported by IoT:Poker and represented by the **teal** paths in Figure 2. The third row concerns a *compartmentalisation* property (a query `isolated`([A], [B]) checks if the devices in the lists [A] and [B] affect each other): there must be no interaction between `bdgReader1` and the devices `pump1` and `elevator`. In

Table I
ANALYSING THE BAS WITH IOT:POKER.

Query	Result	Reason & witness
reach(prsSens1, light1)	✓	prsSens1; light1
reach(smoke1, elevator)	✓	smoke1; fireMngr; elevator
isolated([bdgReader1], [pump1, elevator])	✓	
onlyReachedBy(elevator, [button1, button2, smoke1, smoke2, fireMngr])	✓	
reachOnly(lock1, [light1, prsSens1, log])	✗	lock1; prsSens1; light2

fact, as a security property, we want the elevator and water pumps to be completely isolated from uncorrelated devices like badge readers. This is easily verified by IOT:POKER, because there is no path that violates the query. The fourth row is about the *integrity* of the elevator, whose behaviour should only depend on alarm buttons, smoke sensors, and the fire alarm manager (a query onlyReachedBy(A, [B]) determines if the information affecting device A has been only generated by the devices in the list [B]). This security property is satisfied: the only paths reaching the elevator in the information flow graph are those produced by the intended devices, i.e. those in **teal** and **cyan** in Figure 2. The last row regards *confidentiality* (a query reachOnly(A, [B]) checks whether the information published by the device A can only affect devices in the list [B]). It states that the only devices that can receive information from a lock are the log service, the light bulb, and the presence sensor on the same floor as the lock. IOT:POKER finds out that this property is not satisfied, as a compromised light bulb can receive messages intended for other light bulbs. The problematic flow is the **red** path in Figure 2 showing that information from lock1 can reach light2 on the second floor.

This violation occurs because the policy associated with light bulbs is overly permissive (see Figure 3). The policy has two statements: the first allows the bulb to connect to the Cloud using a freely chosen identifier and to receive messages on any topic since we use the wildcard '$*$'; the second statement permits it to subscribe to topic filters of the form phAC/floor?/dtdMovement/${iot : ClientId}, where '?' stands for any single character and ${iot : ClientId} for the identifier chosen during connection. Note that, to receive messages from the presence sensor, the light bulb 1 must connect using the right identifier, namely, light1. However, this choice is not imposed on the light bulb, which can connect with any other client id when compromised. Indeed, a compromised light bulb can connect to the broker using the MQTT wildcard # as its client id. Consequently, once the policy is instantiated, the light bulb can subscribe to the topic phAC/floor1/dtdMovement/# to receive messages from the presence sensors of both building floors, violating our security property. Our tool can easily help system designers spot such subtle misconfiguration.

The online repository [9] contains our BAS's complete IoT configuration consisting of 17 devices, 20 certificates, and 15 IoT policies of ∼20 loC each, and the instruction to build the information flow graph and answer our queries.

```
{ "Effect":"Allow",
  "Action":"[ iot:Connect, iot:Receive ]",
  "Resource":"*" },
{ "Effect":"Allow",
  "Action":"iot:Subscribe",
  "Resource":"phyAC/floor?/dtdMovement/${iot:ClientId}" }
```

Figure 3. The policy of the light bulbs in the BAS.

IV. A FORMAL MODEL OF IOT ACCESS POLICIES

Here, we take AWS IoT Core as a reference implementation of an IoT-Cloud infrastructure, and we formalize the MQTT broker component and the policy evaluation procedure. We focus on the core features of MQTT that concern the exchange of messages, leaving out some advanced features such as retain and last will messages, and those concerning quality of service and the authentication of devices. This allows us to have a minimal model that captures the essence of the protocol. Similarly, we model the core of the AWS policy language and of its evaluation process, focusing on the interplay between MQTT and the authorization mechanism. In doing so, we clarify the counter-intuitive semantics of AWS and MQTT wildcards and variable substitution in policy evaluation.

Our presentation follows a bottom-up approach.

Definition 1 (Resource)**.** *A resource ρ is either a client id, a topic, or a topic filter. Let Λ be the alphabet of alphanumerical characters enriched with the forward-slash character '/'. Topics are strings over Λ, while client ids and topic filters are strings over Λ, further extended with the MQTT wildcards '+' and '#'.*

We denote with $\mathbb{I}$, $\mathbb{T}$, and $\mathbb{TF}$ the set of client ids, topics, and topic filters, respectively. Note that these sets have a non-empty intersection so as to model features of the language like substitutions of client ids for $clientId variable and matching topics with topic filters. MQTT wildcards are treated as common characters in client ids. However, their semantics is the one prescribed by the MQTT protocol when occurring in topic filters. Note indeed that, although client ids and topic filters have the same syntax, a topic filter tf can be interpreted as a regular expression $tf^{\texttt{mqtt}}$ where the character '#' is interpreted as any sequence of characters in Λ, while the character '+' as any sequence of characters different from '/' in Λ. The language of $tf^{\texttt{mqtt}}$, denoted as $\mathcal{L}(tf^{\texttt{mqtt}})$, contains the topics that are matched by the topic filter tf. For example, the topic filter abc/+/f/# can be translated into the regular expression abc/[a−zA−Z0−9]*/f/[a−zA−Z0−9/]*, where [a−zA−Z0−9] denotes alphanumerical characters, [a−zA−Z0−9/]is the same but also includes /, and x* represents any sequence of characters in x. Note that, e.g., the topic abc/de/f/g/h is in $\mathcal{L}(\texttt{abc/+/f/\#}^{\texttt{mqtt}})$.

In a policy, resources are specified by resource expressions, strings representing both MQTT topics, topic filters, and client ids. These strings may contain AWS wildcards to compactly represent a set of resources, and the variable ${iot : ClientId} (that we write more compactly as $clientId).

Definition 2 (Resource Expression)**.** *A resource expression* Re *for a given resource is a string built on the same alphabet of the resource extended with the variable* $\underline{\texttt{\$clientId}}$ *and with the AWS wildcards '?' and '$*$'. A resource expression is* ground *if it does not contain* $\underline{\texttt{\$clientId}}$*.*

Given a resource expression Re and a string v, we denote with $\texttt{Re}[^{v}/_{\underline{\texttt{\$clientId}}}]$ the resource expression where every occurrence of $\underline{\texttt{\$clientId}}$ is replaced with v. For example, $\texttt{dtdMovement?}/\underline{\texttt{\$clientId}}[^{\#}/_{\underline{\texttt{\$clientId}}}]$ is $\texttt{dtdMovement?}/\#$. A ground resource expression Re can be interpreted as a regular expression $\texttt{Re}^{\texttt{aws}}$ where '?' stands for any character in Λ extended with the MQTT wildcards, and '$*$' for any sequence of such characters. The language $\mathcal{L}(\texttt{Re}^{\texttt{aws}})$ contains the resources that are matched by the resource expression Re. For example, the resource expression for the topic filter $\texttt{?bc}/+/*$, which uses the AWS wildcards '?' and '$*$' (and the MQTT wildcard '+'), can be translated into the regular expression $[\texttt{a}-\texttt{zA}-\texttt{Z0}-\texttt{9}/+\#]\texttt{bc}/+/[\texttt{a}-\texttt{zA}-\texttt{Z0}-\texttt{9}/+\#]^*$. Note that the topic filter $\texttt{abc}/+/\texttt{f}/\texttt{g}/\#$ is in $\mathcal{L}(\texttt{?bc}/+/*^{\texttt{aws}})$.

We now define IoT policies: a policy establishes which client ids can connect and which topics they can publish and/or subscribe to.

Definition 3 (Policy)**.** *A policy P is a set of statements $s = (\varepsilon, \alpha, \mathcal{R})$ where*

- $\varepsilon \in \{\texttt{Allow}, \texttt{Deny}\}$ *is the effect of the statement;*
- $\alpha \in \mathbb{A} = \{\texttt{Con}, \texttt{Pub}, \texttt{Rec}, \texttt{Sub}\}$ *is the IoT action to which the effect applies;*
- $\mathcal{R}$ *is a set of resource expressions.*

The definition of ground and of substitution is naturally extended to policies.

After policies, we introduce resource permissions and formalize their evaluation. Intuitively, upon message reception, the broker interrogates the access control system to check if the action requested by the message is allowed. To do that, the access control system checks whether there exists a policy statement s in P forbidding the request, i.e., with the same action of the request, with effect Deny, and with the resource ρ matched by a resource expression in $\mathcal{R}$ when aws wildcards are considered. The request is rejected and the evaluation stops, if a match is found. Otherwise, the system looks for a statement s that explicitly allows the request. If there is at least one match, the request is accepted. If no matching statement is found, the request is denied (*default deny*).

Definition 4 (Permission)**.** *A permission is a pair (α, ρ) where α is an IoT action to be performed over a resource ρ. A permission (α, ρ) is granted by the* ground *policy P, in symbols $P \models (\alpha, \rho)$, if and only if both the following hold:*

- $\{(\texttt{Allow}, \alpha, \{\texttt{Re}\} \cup \mathcal{R})\}$ *exists in P with $\rho \in \mathcal{L}(\texttt{Re}^{\texttt{aws}})$;*
- *for every $(\texttt{Deny}, \alpha, \mathcal{R}) \in P$, no $\texttt{Re} \in \mathcal{R}$ satisfies $\rho \in \mathcal{L}(\texttt{Re}^{\texttt{aws}})$.*

To configure IoT Core, users define certificates to identify the entities authorized to connect to the broker. These certificates are associated with specific policies. Additionally, we assume the existence of a mapping from devices to their certificates. Formally:

Definition 5 (Configuration)**.** *A configuration is a 5-tuple $(\mathbb{C}, \mathbb{P}, \varphi, \mathbb{D}, k)$ where*

- $\mathbb{C}$ *is a set of certificates;*
- $\mathbb{P}$ *is a set of IoT policies;*
- $\varphi \colon \mathbb{C} \to \mathcal{P}(\mathbb{P})$ *is a function that associates a certificate with its policies;*
- $\mathbb{D}$ *is a set of devices;*
- $k \colon \mathbb{D} \to \mathcal{P}(\mathbb{C})$ *is a function that associates a device with its certificates.*

We write $\Phi(c) = \bigcup_{P \in \varphi(c)} P$ for the union of c's policies.

Example 1. *Consider the BAS of Section III, the certificates of both the light bulbs are associated with the policy of Figure 3, i.e., $\varphi(c_{\texttt{light1}}) = \varphi(c_{\texttt{light2}})$.*

We are now ready to formally define the evolution of an IoT system according to the interaction between devices and the broker: devices send MQTT messages over established connections (typically TCP), requesting specific operations on designated resources. More precisely, given a configuration, we define a labeled transition system (LTS): its states represent the internal state of the broker, its labels are MQTT messages, and the transitions describe how the broker state changes upon reception of messages. We start from the broker state, which is characterized by the active (TCP) connections, a mapping from connections to their corresponding policies, and a mapping from topic filters to the connections of the subscribed devices.

Definition 6 (Broker State)**.** *Given a configuration $(\mathbb{C}, \mathbb{P}, \varphi, \mathbb{D}, k)$, a broker state is a triple $\sigma = (\mathbb{L}, p, \varsigma)$ where*

- $\mathbb{L}$ *is a set of active connections;*
- $p \colon \mathbb{L} \to \mathbb{P}$ *maps connections to their policies;*
- $\varsigma \colon \mathbb{TF} \to \mathcal{P}(\mathbb{L})$ *maps topic filters to their subscribed connections.*

In the following, we assume a function $dev \colon \mathbb{L} \to \mathbb{D}$ to associate each connection ℓ with its corresponding device. Moreover, when referring to p and ς, we represent a function as a set of pairs $x \mapsto y$. Moreover, we use a *function redefinition* operation $f \triangleleft [x \mapsto y]$ such that the result returns y when the input is x and behaves like f for all other inputs.

The labels of the LTS are *MQTT message*:

Definition 7 (MQTT Message)**.** *Let ℓ be a connection, an MQTT message over ℓ is a pair $m = (\ell, a)$ where a is either*

- $\texttt{Con}(c, id)$ *- a connection request;*
- $\texttt{Sub}(tf)$ *- a subscription request;*
- $\texttt{Disc}$ *- a disconnection request;*
- $\texttt{UnSub}(tf)$ *- a unsubscription request*
- $\texttt{Pub}(t)$ *- a publication request.*

Given a configuration $(\mathbb{C}, \mathbb{P}, \varphi, \mathbb{D}, k)$, a transition $\sigma \xrightarrow{m} \sigma'$ specifies how the broker state σ evolves into σ' upon the reception of the MQTT message m. Transitions are defined

by the rules of Figure 4 if some apply, or by the idle move $\sigma \xrightarrow{m} \sigma$ otherwise. The rule (CON) states that when receiving a connection request, the broker authenticates the device association with the certificate; instantiates the policy of the provided certificate with the declared client id; asks the access control system if the obtained ground policy permits the connection; finally, if the connection is permitted, it updates the set of connected devices and records the association of the connection with the policy. The rule (SUB) states that when receiving a subscription request, the broker retrieves the policy associated with the connection; asks the access control system if the obtained ground policy permits the subscription; finally, if the permission is granted, it updates the set of subscribers to the topic filter by adding the connection. The rule (DISC) simply states that when receiving a disconnection request, the broker removes all the occurrences of the connection from its state. Finally, (UNSUB) manages unsubscription requests by removing from the ς function the binding between the connection and the specific topic filter, if present. All other messages do not cause a change in the broker states, either because it is not part of its intended semantics (e.g. `Pub` messages), or the request is denied by the access control mechanism, or the message is discarded (e.g. if a subscription request arrives from a device that is not connected).

Definition 8 (Broker Execution)**.** *A broker execution π is a finite sequence of transitions*

$$\sigma_0 \xrightarrow{m_1} \sigma_1 \xrightarrow{m_2} \sigma_2 \xrightarrow{m_3} \dots \xrightarrow{m_n} \sigma_n$$

We say that π is feasible *if σ_0 is the initial* empty *state $(\emptyset, \emptyset, \{tf \mapsto \emptyset \mid tf \in \mathbb{TF}\})$.*

Hereafter, we denote with $\pi \odot \pi'$ the concatenation of executions, which is only defined when the last element of π coincides with the first one of π' (in the resulting execution, we omit the first broker state of π' to avoid duplications).

Since we are interested in analyzing how information propagates between devices through the broker, we introduce communication triples $d \overset{t}{\Rightarrow} d'$, meaning that d has communicated some information to d' through the topic t. We compose communication triples into communication traces.

Definition 9 (Communication Trace)**.** *A communication trace ϖ from d_0 to d_n is a finite sequence of communication triples*

$$d_0 \overset{t_1}{\Longrightarrow} d_1 \overset{t_2}{\Longrightarrow} d_2 \overset{t_3}{\Longrightarrow} \dots \overset{t_n}{\Longrightarrow} d_n.$$

We extend $\odot$ to communication traces and to sets, with the proviso that $\emptyset$ is the identity element.

We now show how to derive communication traces from broker executions. First, we introduce the auxiliary notion of the set of devices that can receive messages published over a topic t when the broker is in a state σ. Formally,

Definition 10. *Given a broker state $\sigma = (\mathbb{L}, p, \varsigma)$ and a topic t, the set $Rec(\sigma, t) \subseteq \mathbb{D}$ is defined as*

$$Rec(\sigma, t) = \bigcup_{tf \in \mathbb{TF}\downarrow_t} \{dev(\ell) \mid \ell \in \varsigma(tf) \text{ and } p(\ell) \models (\texttt{Rec}, t)\}$$

where $\mathbb{TF}\downarrow_t = \{tf \mid t \in \mathcal{L}(tf^{\texttt{mqtt}})\}$.

Definition 11 (Traces of an Execution)**.** *The set $Tr(\pi)$ of the communication traces of an execution π is defined as*

$$Tr(\pi) = \begin{cases} \{d \overset{t}{\Rightarrow} d' \mid d = dev(\ell) \text{ and } d' \in Rec(\sigma, t)\} & \text{if } \pi = \sigma \xrightarrow{(\ell, \texttt{Pub}(t))} \sigma' \text{ with } \sigma = (\mathbb{L}, p, \varsigma) \text{ and } p(\ell) \models (\texttt{Pub}, t) \\ Tr(\sigma \xrightarrow{m} \sigma') \odot Tr(\pi') & \text{if } \pi = \sigma \xrightarrow{m} \sigma' \odot \pi' \\ \emptyset & \text{otherwise} \end{cases}$$

Example 2. *The light bulb* `light2` *can connect with # its client id by creating a connection ℓ and by sending the MQTT message $(\ell, \texttt{Con}(c_{\texttt{light2}}, \#))$. The policy of Figure 3 is instantiated as the following P:*

$$\{(\texttt{Allow}, \texttt{Con}, \{*\}), (\texttt{Allow}, \texttt{Rec}, \{*\}), (\texttt{Allow}, \texttt{Sub}, \{\texttt{phAC/floor?/dtdMovement/\#}\})\}$$

The condition $P \models (\texttt{Con}, \#)$ holds because $\# \in \mathcal{L}(*^{\texttt{aws}})$, i.e. the chosen id is included in the resource expression of the first allow statement (and there is no deny statement). Note that the instantiated policy allows the device to subscribe to the topic filter `phAC/floor1/dtdMovement/#`*, where the character '#' of the client id is interpreted as an MQTT wildcard.*

Attacker Model

We assume that the cloud infrastructure, the broker, and the authorization mechanisms are trusted and safe from tampering. On the contrary, an attacker can take full control of a device d, thus gaining the capability of authenticating with all the certificates in $k(d)$. The compromised device d ignores its original scope and sends arbitrary messages to the broker. The device can also connect with multiple client ids to maximize the number of topics over which it sends and from which it receives information. Given a compromised device dev, we analyze three security properties:

Confidentiality: determine whether d can gain some private information from another device d', even indirectly, through a communication trace from d' to d in $Tr(\pi)$ for some feasible execution π;

Integrity: determine whether d can influence the behaviour of another critical device d', even indirectly, by sending some information through a communication trace from d to d' in $Tr(\pi)$ for some feasible execution π;

Isolation: determine that there are no possible interactions between d and another specific device d', i.e., no communication traces allow them to interact with each other.

V. CHECKING INFORMATION FLOW

We present now how to check the information flow properties of an IoT configuration. The underlying idea is to build a bipartite graph whose nodes represent certificates (the connected devices) and topics: there is an arc from a certificate c to a topic t if the policy allows c to publish on t; vice versa there is an arc from t to c if the policy allows c to subscribe and receive messages on t. Note that this graph is a static

$$\text{(Con)}\ \frac{m = (\ell, \texttt{Con}(c, id)) \quad c \in k(dev(\ell)) \quad P = \Phi(c)[\,^{id}/_{\underline{\texttt{\$clientId}}}] \quad P \models (\texttt{Con}, id)}{(\mathbb{L}, p, \varsigma) \xrightarrow{m} (\mathbb{L} \cup \{\ell\}, p \triangleleft [\ell \mapsto P], \varsigma)}$$

$$\text{(Sub)}\ \frac{m = (\ell, \texttt{Sub}(tf)) \quad \ell \in \mathbb{L} \quad p(\ell) \models (\texttt{Sub}, tf)}{(\mathbb{L}, p, \varsigma) \xrightarrow{m} (\mathbb{L}, p, \varsigma \triangleleft [tf \mapsto \varsigma(tf) \cup \{\ell\}])} \qquad \text{(UnSub)}\ \frac{m = (\ell, \texttt{UnSub}(tf)) \quad \ell \in \mathbb{L}}{(\mathbb{L}, p, \varsigma) \xrightarrow{m} (\mathbb{L}, p, \varsigma \triangleleft [tf \mapsto \varsigma(tf) \setminus \{\ell\}])}$$

$$\text{(Disc)}\ \frac{m = (\ell, \texttt{Disc}) \quad \mathbb{L}' = \mathbb{L} \setminus \{\ell\}}{(\mathbb{L}, p, \varsigma) \xrightarrow{m} (\mathbb{L}', \{\ell' \mapsto p(\ell') \mid \ell' \in \mathbb{L}'\}, \{tf \mapsto \varsigma(tf) \cap \mathbb{L}' \mid tf \in \mathbb{TF}\})}$$

Figure 4. Broker state evolution.

representation of the communications that may arise in some feasible execution of the IoT system.

It is convenient to introduce some auxiliary definitions for constructing our graphs. For each certificate c, we define $\mathbb{T}_c^O$ as the set of topics over which the devices that can connect with certificate c may send messages. Similarly, $\mathbb{T}_c^I$ collects topics over which c may receive messages.

Definition 12 (I/O Sets)**.** *Given a configuration and a certificate c, its input and output sets $\mathbb{T}_c^I$ and $\mathbb{T}_c^O$ are set of topics defined as:*

$$\begin{aligned}
\mathbb{T}_c^O = \{t \mid\ & \Phi(c)[\,^{id}/_{\underline{\texttt{\$clientId}}}] \models (\texttt{Con}, id) \\
& \text{and } \Phi(c)[\,^{id}/_{\underline{\texttt{\$clientId}}}] \models (\texttt{Pub}, t), \text{ for some } id\} \\
\mathbb{T}_c^I = \{t \mid\ & \Phi(c)[\,^{id}/_{\underline{\texttt{\$clientId}}}] \models (\texttt{Con}, id), \\
& \Phi(c)[\,^{id}/_{\underline{\texttt{\$clientId}}}] \models (\texttt{Rec}, t), \\
& \Phi(c)[\,^{id}/_{\underline{\texttt{\$clientId}}}] \models (\texttt{Sub}, tf) \\
& \text{and } t \in \mathcal{L}(tf^{\texttt{mqtt}}), \text{ for some } id \text{ and } tf\}
\end{aligned}$$

Roughly speaking, the conditions on the first set require that c can connect with client id id and can publish over t. The conditions on the second sets, instead, require c to be able to connect with client id id, to subscribe to the topic filter tf, and that topic t belongs to the language of tf.

Example 3. *Consider the following topic $t =$ $\texttt{phAC/floor1/dtdMovement/light1}$ and the light bulb $\texttt{light2}$. It holds that $t \in \mathbb{T}^I_{c_{\texttt{light2}}}$ because: (i) the light bulb can connect to the broker with client id #, and the substitution gives the policy of Example 2; (ii) the ground policy allows the client to receive messages on t and to subscribe to the topic filter $tf = \texttt{phAC/floor1/dtdMovement/\#}$; and (iii) the topic t is a string in the language of tf when considering MQTT wildcards.*

We now define information flow graphs of configurations:

Definition 13 (Information Flow Graph)**.** *Given a configuration, its information flow graph is a bipartite directed graph $G = (\mathbb{C} \cup \mathbb{T}, E)$ where $\mathbb{C}$ is the set of certificates, $\mathbb{T}$ of all possible topics, and $E \subseteq (\mathbb{C} \times \mathbb{T}) \cup (\mathbb{T} \times \mathbb{C})$ of the arcs defined as*

$$E = \{(c, t) \mid t \in \mathbb{T}_c^O\} \cup \{(t, c) \mid t \in \mathbb{T}_c^I\}$$

An arc of G represents a communication permitted by the configuration. Following a path of G, we can track all the possible device interactions, including indirect ones. We say that there is a (possible) information flow from c to c' if c' is reachable from c in the information flow graph, meaning that information produced by c may affect c'.

Example 4. *A path violating the requirement of the last row of Table I is $c_{\texttt{lock1}} \to t_1 \to c_{\texttt{prsSens1}} \to t_2 \to c_{\texttt{light2}}$ where the topics t_1 and t_2 are*

$$t_1 = \texttt{phAC/floor1/prsSens1/enable}$$
$$t_2 = \texttt{phAC/floor1/dtdMovement/light1}.$$

The following theorem ensures that information flow graphs is both a correct and complete representation of the semantics introduced in the previous section:

Theorem 1. *Given a configuration and its information flow graph G, it holds that:*

1) *if G contains both the arcs (c, t) and (t, c') then for each pair of devices d, d' such that $c \in k(d)$ and $c' \in k(d')$ a feasible execution π* exists *with $d \overset{t}{\Rightarrow} d' \in Tr(\pi)$; and*
2) *if d, d' exists such that $c \in k(d)$ and $c' \in k(d')$, and there is a feasible execution π such that $d \overset{t}{\Rightarrow} d' \in Tr(\pi)$, then G contains both the arcs (c, t) and (t, c').*

Note that certificates (which are associated with access control policies) suffice to analyze communication among devices: we can check if the device d can send a message to d' by considering communication among the certificates in $k(d)$ and $k(d')$. This also means that we do not need to recompute the information flow graph to handle connections and disconnections of devices.

As a final remark, recall that we do *not* identify a client with its client id. This is because only the association with the certificate is authenticated by the Cloud, whereas the client id is freely selected by the device. A compromised device might exploit this flexibility by using multiple ids or MQTT wildcards to maximize the attack surface.

The following result establishes that device reachability through multi-step communication traces in the semantics is consistent with possible information flows in the graph. This ensures that we can use the graph for analyzing the security

properties of confidentiality, integrity, and isolation of our attacker model.

Corollary 1. *Given a configuration and its information flow graph G, for each pair of certificates c and c' it holds that:*

1) *if G has an information flow from c to c' then for each d and d' with $c \in k(d)$ and $c' \in k(d')$, a feasible execution π* exists *such that a trace from d to d' is in $Tr(\pi)$; and*
2) *if d, d' and π feasible exist such that $c \in k(d)$, $c' \in k(d')$, and $Tr(\pi)$ contains a trace from d to d', then G has an information flow from c to c'.*

In general, the set of topics may be infinite, so the information flow graph may contain an infinite set of arcs. Clearly, building it is infeasible in practice. To address this issue, we build a symbolic version where sets of topics are represented by logical formulas.

We first define predicates representing the conditions under which a permission may be granted during some execution.

Definition 14 (Permission Predicate)**.** *Given a 4-tuple (c, id, α, ρ) where c is a certificate, α is an IoT action, ρ is a resource, we let the* permission predicate $\psi_{(c,id,\alpha,\rho)}$ *to be the defined as follows, where P is $\Phi(c)[^{id}/_{\underline{\texttt{\$clientId}}}]$:*

$$\psi_{(c,id,\alpha,\rho)} := \Big(\bigvee_{(\texttt{Allow},\alpha,\mathcal{R}) \in P} \bigvee_{\texttt{Re} \in \mathcal{R}} \rho \in \mathcal{L}(\texttt{Re}^{\texttt{aws}}) \Big) \wedge \neg \Big(\bigvee_{(\texttt{Deny},\alpha,\mathcal{R}) \in P} \bigvee_{\texttt{Re} \in \mathcal{R}} \rho \in \mathcal{L}(\texttt{Re}^{\texttt{aws}}) \Big)$$

We introduce the following auxiliary predicates $\mathcal{I}_c(t)$ and $\mathcal{O}_c(t)$, mimicking input and output sets in a symbolic setting.

Definition 15 (I/O Predicates)**.** *Given an IoT configuration I and a certificate c, its input and output predicates $\mathcal{I}_c$ and $\mathcal{O}_c$ are defined as:*

$$\mathcal{O}_c(t) = \exists\, id\, .\psi_{(c,id,\texttt{Pub},t)} \wedge \psi_{(c,id,\texttt{Con},id)};$$
$$\mathcal{I}_c(t) = \exists\, id, tf .\psi_{(c,id,\texttt{Sub},tf)} \wedge \psi_{(c,id,\texttt{Rec},t)} \wedge \psi_{(c,id,\texttt{Con},id)} \wedge t \in \mathcal{L}(tf^{\texttt{mqtt}})$$

The symbolic version of the information flow graph is intuitively defined as a graph with a node for each certificate, and an additional node for each pair of certificates that can communicate. More precisely, for each pair of certificates c and c', we build a logic formula $F_{c,c'}$ that is true if and only if c can publish a message that can be read by (devices authenticated with) c'. If the formula is valid, we add a node $F_{c,c'}$, together with arcs from c to $F_{c,c'}$ and from $F_{c,c'}$ to c' (see the next section for the algorithm to compute the symbolic graph).

Definition 16 (Symbolic Information Flow Graph)**.** *Given an IoT configuration I, its symbolic information flow graph is $G_s = (N, E)$:*

$$N = \mathbb{C} \cup \{\ F_{c,c'} \mid c, c' \in \mathbb{C} \textit{ and } F_{c,c'} \textit{ is true}\}$$
$$E = \{(c, F_{c,c'}) \mid c, c' \in \mathbb{C}\} \cup \{(F_{c,c'}, c') \mid c, c' \in \mathbb{C}\}$$

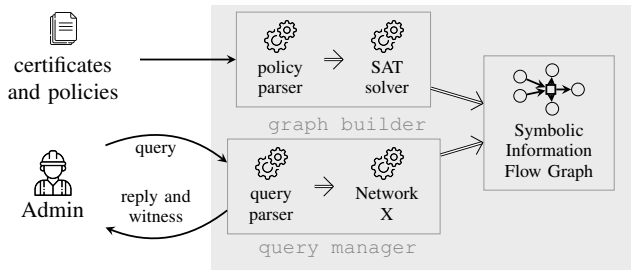

Figure 5. Overview of IoT:Poker.

where for each c and c', $F_{c,c'} = \exists t.\mathcal{O}_c(t) \wedge \mathcal{I}_{c'}(t)$ is a logic formula over the (first order) theory of regular expressions.

We say that there is a (possible) symbolic information flow from c to c' if c' is reachable from c in the symbolic information flow graph G_s.

Example 5. *The red path in Figure 2 is $c_{\texttt{lock1}} \rightarrow F_{c_{\texttt{lock1}},c_{\texttt{prsSens1}}} \rightarrow c_{\texttt{prsSens1}} \rightarrow F_{c_{\texttt{prsSens1}},c_{\texttt{light2}}} \rightarrow c_{\texttt{light2}}$. The violating path of Example 4 is one of its instances where the topics t_1 and t_2 satisfy the formula $F_{c_{\texttt{lock1}},c_{\texttt{prsSens1}}}$ and $F_{c_{\texttt{prsSens1}},c_{\texttt{light2}}}$.*

The following theorem establishes that the symbolic information flow graph provides a sound and complete representation of the actual information flow graph.

Theorem 2. *Given a configuration, there is an information flow from c to c' if and only if there is a symbolic information flow from c to c'.*

The next section presents the implementation of our tool for checking information flow in IoT systems. However, we highlight a crucial point: the symbolic information flow graph cannot be computed solely through operations on regular languages, as one might initially assume, given that both AWS and MQTT wildcards can be described using regular language operators. The issue arises from the substitution of the variable $\underline{\texttt{\$clientId}}$ within the policy rules that complicates the task. In particular, $\mathbb{T}_c^I$ and $\mathbb{T}_c^O$ are not always regular languages, as shown by the following example.

Example 6. *Consider a certificate c with*

$$\Phi(c) = \{(\texttt{Allow}, \texttt{Con}, *), (\texttt{Allow}, \texttt{Pub}, \underline{\texttt{\$clientId}}/\underline{\texttt{\$clientId}})\}.$$

The output set of topics over which c can publish $\mathbb{T}_c^O = \{\omega/\omega \mid \omega \in (\Lambda \cup \{/\})^*\}$ is not *a regular language, as it contradict the pumping lemma [14].*

VI. Implementation and Evaluation

We implemented our verification approach in IoT:Poker [9], an open-source AWS IoT-policy analyzer written in Python. Its architecture is in Figure 5: the tool takes a configuration as input, parses the policies via the library Policy Universe [15] and uses the SMT solver cvc5 [16] to build the symbolic information flow graph. Finally, a query

Algorithm 1 Symbolic Information Flow Graph Construction.

Input: a configuration $(\mathbb{C}, \mathbb{P}, \varphi, \mathbb{D}, k)$
Output: its symbolic information flow graph

1: $(N, E) \leftarrow (\mathbb{C}, \emptyset)$
2: **for all** $(c, c') \in \mathbb{C} \times \mathbb{C}$ **do**
3: build $F_{c,c'}$ as in Definition 16
4: check its satisfiability through Algorithm 2
5: **if** $F_{c,c'}$ is satisfiable **then**
6: $E \leftarrow E \cup \{(c, F_{c,c'}), (F_{c,c'}, c')\}$
7: $N \leftarrow N \cup \{F_{c,c'}\}$
8: **return** (N, E)

Algorithm 2 Witness for the predicate $F_{c,c'}$

Input: two certificates c, c'
Output: a witness $w = (id, id', t, tf)$ or None

1: $S \leftarrow \psi_{(c,id,\texttt{Con},id)} \wedge \psi_{(c',id',\texttt{Con},id')} \wedge \psi_{(c,id,\texttt{Pub},t)}$
2: $\wedge\ \psi_{(c',id',\texttt{Rec},t)} \wedge \psi_{(c',id',\texttt{Sub},tf)}$
3: **if** SMT(S) is unsat **then**
4: **return** None ▷ Early Fail
5: $S_e \leftarrow S \wedge (t = tf)$
6: **if** SMT(S_e) is sat **then**
7: **return** $w \leftarrow (id, id', t, tf)$ ▷ Early Success
8: **for** $n = 1, \ldots, 8$ **do**
9: $S_h \leftarrow S \wedge t = t_1/t_2/\ldots/t_n$
10: $S'_h \leftarrow tf \in \mathcal{L}((t_1|+)/\ldots/(t_n|+))$
11: **for** $m = 1, \ldots, n-1$ **do**
12: $S'_h \leftarrow S'_h \vee tf \in \mathcal{L}((t_1|+)/\ldots/(t_m|+)/\#)$
13: **if** SMT($S_h \wedge S'_h$) is sat **then**
14: **return** $w \leftarrow (id, id', t, tf)$
15: **return** None

manager answers users' requests by checking reachability in the graph via the NetworkX library [17].

The following first describes how the symbolic information flow graph is built, and its the main technical challenges; then, clarifies what queries can be performed on the resulting graph; finally, provides an experimental evaluation of the scalability of IoT:POKER.

A. Building the graph

Given a configuration $(\mathbb{C}, \mathbb{P}, \varphi, \mathbb{D}, k)$, initially, we start from a symbolic graph that has a node for each certificate $c \in \mathbb{C}$ and no arcs. Then, we iterate over each pair of certificates c and c' of $\mathbb{C}$, building the predicate $F_{c,c'}$ of Definition 16, encoding it into an SMT formula (a simple form of Skolemization), and invoking the solver: if the solver finds that the formula is satisfiable, then we add a node for $F_{c,c'}$ and a pair of arcs from c to $F_{c,c'}$ and from $F_{c,c'}$ to c'. This procedure is detailed in Algorithm 1.

The core challenge is efficiently deciding the SMT problem, which amounts to finding two client ids (id, id'), a topic (t) and a topic filter (tf), such that the predicates $\mathcal{O}_c$ and $\mathcal{I}_{c'}$ hold, i.e. satisfying the formula $\Psi(id, id', t, tf)$ below, or proving it impossible.

$$\Psi(id, id', t, tf) = \psi_{(c,id,\texttt{Con},id)} \wedge \psi_{(c',id',\texttt{Con},id')} \wedge \psi_{(c,id,\texttt{Pub},t)} \wedge \psi_{(c',id',\texttt{Rec},t)} \wedge \psi_{(c',id',\texttt{Sub},tf)} \wedge t \in \mathcal{L}(tf^{\texttt{mqtt}})$$

The predicates can be encoded by resorting to the strings and regular expressions theories, but the presence of two different kinds of wildcards (AWS and MQTT) in the policies makes the problem challenging in practice: the solver would take hours and hours of computation to check the satisfiability of a single predicate $F_{c,c'}$, if a naive encoding of the formula is used. For solving this problem, we implement an effective strategy to invoke the solver in Algorithm 2, which is divided in three stages. The first two stages are heuristics for early failure and success, while the last one is a complete decision procedure, optimised for dealing with the potency of wildcards by taking advantage of the peculiar structure of the $F_{c,c'}$ formulas.

Our heuristics are based on checking weakened and strengthened versions of Ψ, where the constraint $t \in \mathcal{L}(tf^{\texttt{mqtt}})$ is approximated by simpler conditions. In the first stage, the algorithm considers a set of conditions that are necessary but not sufficient for guaranteeing communication between c and c'. The constraint $t \in \mathcal{L}(tf^{\texttt{mqtt}})$ is removed and the solver is asked to simply determine if there exist some assignment that satisfies at least the $\psi_{\_}$ predicates of Ψ. If this is not the case, then communication is trivially impossible, and an *early fail* occurs (line 4). Note that these conditions are not sufficient for communication because we are imposing no relation between the topic filter tf and the topic t. The second stage checks a condition that is sufficient but not necessary for communication: whether c' can subscribe to the same topic t on which c can publish (line 5). Basically, we ignore the MQTT wildcards and run the solver on a version of Ψ where $t = tf$ substitutes $t \in \mathcal{L}(tf^{\texttt{mqtt}})$. Although simple, these two heuristics greatly increase the performance of our tool, solving most real-world cases.

If neither an early failure nor a solution is found, we adopt the complete resolution strategy (*hard strategy*). This third stage considers Ψ as it is, also taking MQTT wildcard characters into account. Finding an efficient SMT encoding of $t \in \mathcal{L}(tf^{\texttt{mqtt}})$ is made particularly challenging by the fact that tf is not a constant value, but it is rather dynamically computed as a solution of the other constraints. To address this problem, we focus on the peculiar format of IoT resources, exploiting the hierarchical structure of topics $t = t_1/t_2/\ldots/t_n$ and topic filters $tf = tf_1 \,/\, tf_2 \,/\, \ldots \,/\, tf_m$, which have a maximum depth of 8 in AWS. More in detail, we start from the assertions of line 2 about the $\psi_{\_}$ predicates of Ψ, and we add the condition of line 9 stating that t has n levels. Finally, we impose $t \in \mathcal{L}(tf^{\texttt{mqtt}})$ by requiring t and tf to satisfy one of the following:

- the number of levels of t and tf coincides (i.e., $n = m$) and, tf_i matches t_i for each level i, i.e., $tf_i = t_i$ or $tf_i = +$ (line 10, recall that $+$ matches any single level of the topic);

- tf has no more levels than t (i.e., $m \leq n$), tf_i matches t_i for all $i < m$, and the last level of the topic filter is $tf_m = \#$ (line 12, recall that $\#$ matches any remaining sequence of levels $t_m/t_{m+1}/\ldots/t_n$ of the topic t).

For example, the fist case tells us that the topic $t = \texttt{phAC/floor1/prsSens1/enable}$ is included in $\mathcal{L}(tf_1^{\texttt{mqtt}})$ with $tf_1 = \texttt{phAC/+/prsSens1/+}$, because each layer of tf_1 is either equal to the corresponding one of t or it is the wildcard character $+$, which matches both `floor1` and `enable` (as well as any other alphanumerical string). Moreover, the second case tells us that the topic above is also included in $\mathcal{L}(tf_2^{\texttt{mqtt}})$ with $tf_2 = \texttt{phAC/\#}$, because the wildcard character $\#$ matches `floor1/prsSens1/enable` (as well as any other string over the alphabet of alphanumerical characters enriched with the forward-slash character '/').

Our optimisations allow Algorithm 2 to effectively check the satisfiability of the formula $F_{c,c'}$, as shown by our experiments below. An upper bound to the complexity of computing the symbolic graph is given by the following theorem:

Theorem 3 (Complexity)**.** *Let $(\mathbb{C}, \mathbb{P}, \varphi, \mathbb{D}, k)$ be a configuration, and let $f(n)$ be the cost for deciding the satisfiability of quantifier-free predicates of size n using regular expressions and string theories. Then, the time required by Algorithm 1 is at most $O(|\mathbb{C}|^2 \cdot (|\overline{P}| \cdot |\overline{\mathcal{R}}| + f(|\overline{P}| \cdot |\overline{\mathcal{R}}|)))$, where $|\overline{P}|$ and $|\overline{\mathcal{R}}|$ are the size of the largest policy and of the largest resource expression.*

Note that f is at least exponential, independently of the used theories, as the problem subsumes SAT, for which only exponential solutions are known. However, the exponential cost is only on the size of policies and on the number of resource expressions in statements, while the algorithm is polynomial on the size of the network (our experimental results are consistent with this estimation). Noticeably, the size of the network is expected to increase more rapidly than the complexity of the individual policies in practice (as far as policies are adequately designed).

B. Query Manager

IoT:Poker can check four kinds of queries on the symbolic information flow graph. The query $\texttt{reach}(\texttt{A}, \texttt{B})$ verifies whether the information from device `A` affects the one received by device `B`, in symbols $\texttt{A} \rightarrow^* \texttt{B}$ where $\rightarrow^*$ stands for reachability over the information flow graph. The query $\texttt{reachOnly}(\texttt{A}, [\texttt{B}])$ checks whether the information produced by device `A` only affects devices in the list `[B]`, namely, for each certificate `C` we have $\texttt{A} \rightarrow^* \texttt{C}$ if and only if $\texttt{C} \in [\texttt{B}]$. The query $\texttt{onlyReachedBy}(\texttt{A}, [\texttt{B}])$ determines if the information affecting device `A` has been generated by the devices in the list `[B]`: for each certificate `C` we have that $\texttt{C} \rightarrow^* \texttt{A}$ if and only if $\texttt{C} \in [\texttt{B}]$. Finally, the query $\texttt{isolated}([\texttt{A}], [\texttt{B}])$ checks if the devices in `[A]` and `[B]` do not affect each other: for each pair `C` and `C'`, $\texttt{C} \rightarrow^* \texttt{C}'$ implies ($\texttt{C} \notin [\texttt{A}]$ or $\texttt{C}' \notin [\texttt{B}]$) and ($\texttt{C} \notin [\texttt{B}]$ or $\texttt{C}' \notin [\texttt{A}]$).

Each query can be solved by standard search algorithms provided by the NetworkX library, with worst-case performance linearly proportional to the number of edges of the graph. More in detail, $\texttt{reach}(\texttt{A}, \texttt{B})$ and $\texttt{reachOnly}(\texttt{A}, [\texttt{B}])$ are solved by recording the set R of nodes that are reached during a visit of the graph starting from `A`, and by testing $\texttt{B} \in S$ or $S = [\texttt{B}]$, respectively. The same procedure also applies for $\texttt{onlyReachedBy}(\texttt{A}, [\texttt{B}])$, with the reversed direction of the edges. Finally, $\texttt{isolated}([\texttt{A}], [\texttt{B}])$ requires two applications of the search algorithm, where instead of starting from a single node, we start from a given frontier, `[A]` in the first case, and `[B]` on the graph with reverted edges in the second one.

Table II
IoT:Poker scalability over real-world configurations.

devices	nodes avg.	hard strategies avg.	building time (s) min.	building time (s) avg.	building time (s) max.	query time (s) avg.
20	264.6	1.8	0.42	2.87	12.00	0.0060
40	973.0	4.5	2.30	6.28	16.55	0.0086
60	2211.2	5.6	2.93	9.22	23.39	0.0108
80	3903.9	13.6	5.93	16.34	29.08	0.0136
100	6098.1	16.7	7.50	19.46	36.55	0.0167
120	8764.2	21.5	13.58	24.69	41.49	0.0198
140	12080.1	28.7	18.61	31.06	46.63	0.0244
160	15415.9	29.6	20.03	33.01	44.43	0.0280
180	19571.7	37.0	26.61	37.87	52.79	0.0318
200	24209.9	45.8	31.62	46.06	57.34	0.0349
220	29066.2	50.5	32.39	50.93	62.02	0.0393
240	34732.6	59.0	42.11	57.68	62.41	0.0430
258	40019.0	65.0	62.35	62.67	63.53	0.0473

C. Evaluation

We experimentally evaluate the scalability of IoT:Poker on real-world configurations when the IoT configuration's size (number of certificates) grows, using cvc5 as our SMT solver. We conduct all our experiments below on a desktop machine with an i7-10700K processor (3.80GHz) and 32GB RAM, running Windows 11. We consider 258 real-world policies originally implemented for various IoT devices by different manufacturers. These policies are available online in the benchmark of the P-verifier tool [10], and are classified as secure (42) and flawed (216). Using these policies, we performed 30 experiments as follows: (i) we generate 13 configurations with an increasing number of devices from 20 to 258; (ii) each device is randomly associated to a distinct certificate and real-world policy (from the benchmark); (iii) build the symbolic information flow graph of each configuration, measuring its computation time; (iv) interrogate IoT:Poker with 1000 queries concerning reachability between two random devices in each graph, and record the time required to answer.

The results are in Table II, where the first column indicates the size of the IoT configuration (i.e., the number of devices), the second reports the number of nodes of the graph (both certificates and symbolic nodes representing sets of topics), the third one is the number of times the hard strategy was used to determine if two devices communicate, and the last two columns contain the time needed for building the graph and solving the queries. For each configuration size, the table reports the arithmetic mean of the results obtained by the configurations of that size, across the 30 experiments. Moreover, for the graph build time, which is the most expensive part

of the computation, we also mention the fastest and slowest runs. Also considering all the 258 real-world policies of the benchmark, IoT:Poker builds the symbolic graph in less than 64 seconds. Moreover, the time required to build the graph increases linearly as the number of times the algorithm resorts to the hard strategy. The time needed for answering each slot of 1000 queries is less than 0.1 seconds on average. In conclusion, the algorithm scales well in practice up to configurations with more than 250 devices: the one-time effort for building the graph is reasonable, and evaluating queries is almost immediate. Note that most real IoT systems are rarely larger than that.

A version of IoT:Poker also exists, which employs Z3 [18] as SMT solver in place of cvc5. We repeated our experiments with Z3, using the same configurations generated for cvc5. The detailed experimental results are in Appendix B, from which it emerges that cvc5 performs significantly better than Z3.

VII. Related Work

Several papers have investigated the problem of specifying and verifying Cloud-based access policies and information flow in IoT systems. However, to the best of our knowledge, no paper in the literature addresses the verification of information flow properties in Cloud-based IoT access policies.

Backes et al. [5] proposed Zelkova, a tool that statically analyses policies by encoding them into SMT formulas and using off-the-shelf solvers to verify specific requirements. Zelkova can also compare different policies in terms of permission inclusion. With a similar approach, Eiers et al. [6]–[8] use an SMT solver to characterize and reduce the policies' permissiveness quantitatively. More precisely, they aim to automatically repair an overly permissive policy by reducing the number of allowed requests below some predefined threshold. Ze Jin et al. [4] proposed P-Verifier, a tool to detect security flaws in policy configurations. They focus on flaws derived from an incorrect combination of wildcards in the MQTT topic filter and IAM policies that may lead to granting unwanted permissions. Their verification mechanism relies on translating the policies into suitable SMT formulas and checking them with an off-the-shelf solver. All these papers follow an approach similar to ours: using an SMT solver to verify that a policy satisfies some requirements. However, unlike our work, they do not consider information flow but focus on the permissions that a policy grants, aiming at discovering misconfiguration and over-permissive policies. Since the properties they consider differ from ours, their verification mechanism is also different: they analyze each policy in isolation to determine the permissions granted. Consequently, it is not immediately clear how to use their methodology and tools to check information flow properties inside a network of devices, which requires considering several policies. In contrast, our verification mechanism considers all policies at once and all possible device interactions to build the information flow graph. This requires matching the (possibly infinite) sets of topics over which two devices may publish and subscribe, which is challenging because topics may depend on variables, like the client id, and may use wildcards.

Regarding information security in IoT systems, Bastys et al. [19] investigated the problem of securing IoT apps that use the If This Then That paradigm and identified two classes of URL-based attacks that malicious apps can exploit to exfiltrate private information. To mitigate these attacks, they proposed an access control mechanism that classifies apps as private or public, blocking flows from private sources to sensitive sinks. A second proposed countermeasure is a mechanism for tracking information flow within IoT app code, for which they developed a formal model of app execution. Similarly, Yu et al. [20] propose TAPFixer to detect and repair vulnerabilities in the setting of Home Automation. They perform model-checking on the Trigger-Action Programming (TAP) rules of the analyzed system, also taking into account the implicit communication between actuators and sensors through the physical environment. The policy is then fixed by an ad hoc algorithm based on negated-property reasoning. Celik et al. [21] proposed IoTGuard, a dynamic, policy-based enforcement system that monitors the runtime behaviour of IoT apps. The core idea is to instrument the app's code to gather information about its state, which is then passed to a dynamic monitor that detects violations of safety and security policies. Mandalari et al. [22] propose another dynamic approach to automatically classify and block the non-essential network traffic of IoT devices. Some works [23], [24] propose process-algebra models that capture the core aspects of the behaviour of IoT systems, and use formal methods (control-flow analysis, type systems) to prove various security properties. All these papers share a goal similar to ours: ensuring that information flows within an IoT system complies with a given security policy. However, unlike our approach, they assume access to the app's code and ignore access control policies. In contrast, we treat IoT devices as black boxes, over-approximating their behaviour based on the actions their IoT policies allow: we consider an attacker capable of taking full control of a compromised device, subverting its code. As a result, we do not need to re-analyze systems when a device is replaced or its behaviour changes due to an attack, as long as the access policy remains unchanged. Implicitly considering the worst-case scenario for the behaviour of compromised entities, and thus assessing security properties directly on the information flow graphs derived from access control policies (e.g. by investigating reachability), has already proven successful in both role-based and attribute-based access control [25]–[29], as well as in the context of networks [30] and operating systems [31]–[33].

VIII. Discussion

For ease of presentation, the model of Section IV focuses on the core communication features of the MQTT protocol and on the main components of IoT policies provided by AWS IoT Core. Below, we discuss how to extend our formal treatment to model the major additional features and how to keep track of them when checking information flow properties.

To obtain a manageable formal model, we omitted some advanced features of MQTT, such as retain and last will messages, as well as details concerning QoS. We believe that we can easily add these features to our semantics without affecting the validity of our verification methodology. Indeed, even if these features increase the number of ways devices can communicate, they do not allow new interactions among devices that are isolated in our current model: no possible information flow is added or removed.

As for AWS IoT policies, we omitted the advanced features of *things* and *conditions*. Policy administrators can define named virtual devices called *things* in their configurations, which are attached to a set of user-defined attributes, e.g., geographical location. Formally, a thing can be represented as a set of bindings from attribute names to strings which may be used in resource expressions and substituted with their values when the policy is instantiated, similarly to what is done for $\underline{\texttt{\$clientId}}$. For example, in our BAS scenario of Section III one may define the attributes $\underline{\texttt{\$floor}}$ and $\underline{\texttt{\$room}}$, and add the resource expression $\texttt{floornum}\underline{\texttt{\$floor}}/\texttt{roomnum}\underline{\texttt{\$room}}/\texttt{light?}$ in the policy. As a result, two things associating different values to the attributes above may publish on disjoint sets of topics, even if they share the same policy. Our formal model can be updated by considering an additional special case for things in the (CON) rule of Figure 4. Things are associated with connecting devices during certificate authentication, and a specific certificate is either for things or for common devices; hence, the kind of instantiation to use is always clear. If the broker receives a request r from a thing, then the instantiated policy P over which the request is obtained by taking $\Phi(c)$ and applying the substitutions defined by the attributes of the authenticated thing. Updating the definition of the information flow graph and the procedure for building it in Section V and VI is trivial, since things are inherently more constrained than common devices. While we preferred to keep the formal presentation as simple as possible, our tool is capable of dealing with configurations that make use of things, provided their bindings are specified in advance by the user.

An additional feature of policy statements is the optional "Condition" field that further restricts the cases in which the statement is considered when evaluating a request. Conditioning expressions are built using a predefined finite set of operators, keys, and values [12]. Such conditions require mildly extending our model by enriching policy statements of Definition 3 with suitable (decidable) predicates over requests. Then, Definitions 4, 12 and 14 are updated by considering these predicates in conjunction with $\rho \in \mathcal{L}(\texttt{Re}^{\texttt{aws}})$.

Finally, we argue that focusing on AWS IoT Core as a specific instance of IoT-Cloud deployments does not limit the generality of our methodology. On the one hand, focusing on a specific case allows us to concretely put our verification approach to work by targeting real-world configurations. On the other hand, several MQTT brokers support authorization policies for publishing messages and subscribing to topics, e.g., Azure IoT Hub [34] and HiveMQ [35]. These policies are typically written in an RBAC policy language that supports some form of wildcards to specify sets of topics. We are strongly confident that our methodology can be easily adapted to these other MQTT brokers with very few modifications in the technical aspects, e.g., encoding roles through suitable predicates or slightly adjusting the treatment of wildcards. Moreover, the semantics of Figure 4 strictly depend on no specific features of AWS IoT Core and can be adopted for other brokers with very few changes. Similarly, our procedure for the construction of the symbolic information-flow graph of Section V, and for the verification of queries.

IX. Conclusion

We addressed the problem of verifying information flow in IoT access policies, a crucial aspect often overlooked by existing policy analysis tools. We focused on AWS IoT Core's main components, but our approach can be easily adapted to other Cloud platforms supporting the MQTT protocol, as well as to other contexts relying on similar communication models like ROS2 [36]. First, we introduced a formal model of IoT policies and their evaluation during device interactions, ensuring the mathematical basis of our security verification. Then, we built the information flow graph to capture the communication interactions between IoT devices through MQTT topics. We reduce the problem of checking information flow between devices to a graph reachability problem. To enhance practical feasibility, we introduced a symbolic version of the graph, replacing topics with logic formulas to succinctly represent potentially infinite sets of topics. Finally, we implemented our verification mechanism in the IoT:Poker tool and evaluated its scalability on real-world IoT policies.

There are several exciting directions for future work to make our analysis more effective and to foster the adoption of IoT:Poker by practitioners. A first extension involves considering the typical dynamicity of IoT systems, where devices may join and leave. Also, we plan to allow designers to label the certificates with elements of a security lattice and check some predefined queries about flows among labels. Then, we will implement a change-impact analysis to determine how a policy update affects the information flow and if it preserves the security requirements. Finally, we plan to consider scenarios where information flows occur through indirect interaction between devices' sensors and actuators.

References

[1] "Aws iot core," accessed on April 2024. [Online]. Available: https://aws.amazon.com/iot-core/

[2] "Microsoft azure iot hub," accessed on April 2024. [Online]. Available: https://azure.microsoft.com/products/iot-hub/

[3] "Mqtt version 3.1.1," accessed on April 2024. [Online]. Available: https://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html

[4] Z. Jin, L. Xing, Y. Fang, Y. Jia, B. Yuan, and Q. Liu, "P-verifier: Understanding and mitigating security risks in cloud-based iot access policies," in *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security*, 2022.

[5] J. Backes, P. Bolignano, B. Cook, C. Dodge, A. Gacek, K. Luckow, N. Rungta, O. Tkachuk, and C. Varming, "Semantic-based automated reasoning for AWS access policies using SMT," in *Formal Methods in Computer Aided Design*, 2018.

[6] W. Eiers, G. Sankaran, A. Li, E. O'Mahony, B. Prince, and T. Bultan, "Quantifying permissiveness of access control policies," in *Proceedings of the 44th International Conference on Software Engineering*. ACM, 2022.

[7] ——, "Quacky: Quantitative access control permissiveness analyzer," in *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering*, 2023.

[8] W. Eiers, G. Sankaran, and T. Bultan, "Quantitative policy repair for access control on the cloud," in *Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis*, 2023.

[9] "Checking Information Flow in Cloud-based IoT Access Control Policies (Supplementary Material)." [Online]. Available: https://sites.google.com/view/iotpokerpaper

[10] "P-verifier policy benchmark," accessed on June 2024. [Online]. Available: https://github.com/P-Verifier/P-Verifier/tree/master/policy_benchmark

[11] "Amazon web services," accessed on April 2024. [Online]. Available: https://aws.amazon.com/

[12] "Policies and permissions in iam," accessed on April 2024. [Online]. Available: https://docs.aws.amazon.com/IAM/latest/UserGuide/access_policies.html

[13] "Amazon cognito," accessed on April 2024. [Online]. Available: https://aws.amazon.com/cognito/

[14] J. E. Hopcroft and J. D. Ullman, *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley Publishing Company, 1979.

[15] "Policyuniverse," accessed on June 2024. [Online]. Available: https://github.com/Netflix-Skunkworks/policyuniverse

[16] H. Barbosa, C. W. Barrett, M. Brain, G. Kremer, H. Lachnitt, M. Mann, A. Mohamed, M. Mohamed, A. Niemetz, A. Nötzli, A. Ozdemir, M. Preiner, A. Reynolds, Y. Sheng, C. Tinelli, and Y. Zohar, "cvc5: A versatile and industrial-strength SMT solver," in *TACAS (1)*, ser. Lecture Notes in Computer Science, vol. 13243. Springer, 2022, pp. 415–442.

[17] "Networkx," accessed on June 2024. [Online]. Available: https://networkx.org/documentation/stable/index.html

[18] "The z3 theorem prover," accessed on June 2024. [Online]. Available: https://github.com/Z3Prover/z3

[19] I. Bastys, M. Balliu, and A. Sabelfeld, "If this then what? controlling flows in iot apps," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '18, 2018.

[20] Y. Yu, Y. Xu, K. Huang, and J. Liu, "TAPFixer: Automatic detection and repair of home automation vulnerabilities based on negated-property reasoning," in *33rd USENIX Security Symposium*. USENIX Association, 2024.

[21] Z. B. Celik, G. Tan, and P. D. McDaniel, "Iotguard: Dynamic enforcement of security and safety policy in commodity iot," in *NDSS*. The Internet Society, 2019.

[22] A. M. Mandalari, D. J. Dubois, R. Kolcun, M. T. Paracha, H. Haddadi, and D. R. Choffnes, "Blocking without breaking: Identification and mitigation of non-essential iot traffic," *Proc. Priv. Enhancing Technol.*, vol. 2021, no. 4, 2021.

[23] C. Bodei and L. Galletta, "Tracking Sensitive and Untrustworthy Data in IoT," in *Proceedings of the First Italian Conference on Cybersecurity (ITASEC17)*, ser. CEUR Workshop Proceedings, A. Armando, R. Baldoni, and R. Focardi, Eds., vol. 1816. CEUR-WS.org, 2017, pp. 38–52.

[24] M. Balliu, M. Merro, M. Pasqua, and M. Shcherbakov, "Friendly fire: Cross-app interactions in iot platforms," *ACM Trans. Priv. Secur.*, vol. 24, no. 3, Apr. 2021.

[25] M. Bugliesi, S. Calzavara, R. Focardi, and M. Squarcina, "Gran: Model checking grsecurity RBAC policies," in *25th IEEE Computer Security Foundations Symposium*, S. Chong, Ed., 2012.

[26] S. Calzavara, A. Rabitti, and M. Bugliesi, "Compositional typed analysis of ARBAC policies," in *IEEE 28th Computer Security Foundations Symposium*, C. Fournet, M. W. Hicks, and L. Viganò, Eds., 2015.

[27] V. Atluri and W.-K. Huang, "A petri net based safety analysis of workflow authorization models," *J. Comput. Secur.*, vol. 8, no. 2,3, Aug. 2000.

[28] E. Uzun, G. Parlato, V. Atluri, A. L. Ferrara, J. Vaidya, S. Sural, and D. Lorenzi, "Preventing unauthorized data flows," in *Data and Applications Security and Privacy - 31st Annual IFIP WG 11.3 Conference, DBSec 2017, Proceedings*, ser. Lecture Notes in Computer Science, G. Livraga and S. Zhu, Eds., vol. 10359. Springer.

[29] E. Uzun, V. Atluri, J. Vaidya, S. Sural, A. L. Ferrara, G. Parlato, and P. Madhusudan, "Security analysis for temporal role based access control," *J. Comput. Secur.*, vol. 22, no. 6, 2014.

[30] J. D. Guttman and A. L. Herzog, "Rigorous automated network security management," *Int. J. Inf. Sec.*, vol. 4, no. 1-2, 2005.

[31] J. D. Guttman, A. L. Herzog, J. D. Ramsdell, and C. W. Skorupka, "Verifying information flow goals in security-enhanced linux," *J. Comput. Secur.*, vol. 13, no. 1, 2005.

[32] B. S. Radhika, N. V. N. Kumar, R. K. Shyamasundar, and P. Vyas, "Consistency analysis and flow secure enforcement of selinux policies," *Comput. Secur.*, vol. 94, 2020.

[33] L. Ceragioli, L. Galletta, P. Degano, and D. Basin, "Specifying and verifying information flow control in selinux configurations," *ACM Trans. Priv. Secur.*, vol. 27, no. 4, 2024.

[34] "Access control for mqtt clients," accessed on July 2025. [Online]. Available: https://learn.microsoft.com/en-us/azure/event-grid/mqtt-access-control

[35] "Hivemq file role based access control extension," accessed on July 2025. [Online]. Available: https://github.com/hivemq/hivemq-file-rbac-extension

[36] N. V. Pandya, H. Kumar, G. Pillai, and V. Ganapathy, "Decentralized information-flow control for ROS2," in *31st Annual Network and Distributed System Security Symposium, NDSS 2024*, 2024.

APPENDIX A
PROOFS

We start by introducing some useful notation.

Notation 1. *In the following, we write:*

- $\varsigma_\perp$ *for the function* $\{tf \mapsto \emptyset \mid tf \in \mathbb{TF}\}$*;*
- $\sigma_\perp$ *for the initial broker state* $(\emptyset, \emptyset, \varsigma_\perp)$*;*
- $\sigma \to^* \sigma'$ *when* $\sigma \xrightarrow{m_1} \sigma_1 \xrightarrow{m_2} \dots \xrightarrow{m_n} \sigma'$ *for some* $m_1, \dots m_n$ *and* $\sigma_1, \dots \sigma_{n-1}$*;*
- $d \to^* d'$ *when* $d \xrightarrow{t_1} d_1 \xrightarrow{t_2} \dots \xrightarrow{t_n} d'$ *for some* $t_1, \dots t_n$ *and* $d_1, \dots d_{n-1}$*;*

The next three auxiliary results proves that necessary conditions applies to reachability of some given broker states.

Lemma 1. *Let* σ *be a broker state such that* $\sigma_\perp \to^* \sigma = (\mathbb{L}, p, \varsigma)$*, and let* $\ell \in \mathbb{L}$*, then* $p(\ell) = \Phi(c)[^{id}/\underline{\texttt{\$clientId}}]$ *for some* c *and* id *such that* $c \in k(dev(\ell))$ *and* $\Phi(c)[^{id}/\underline{\texttt{\$clientId}}] \models (\texttt{Con}, id)$*.*

Proof. We prove the property by induction on the length of the execution π from $\sigma_\perp$ to σ. The state $\sigma_\perp$ vacuously satisfies the lemma, as no ℓ is in $\mathbb{L} = \emptyset$. Assume $\pi = \pi' \odot (\sigma \xrightarrow{m} \sigma')$ where $\sigma = (\mathbb{L}, p, \varsigma)$ and $\sigma' = (\mathbb{L}', p', \varsigma')$. Then, by induction hypothesis, if $\ell \in \mathbb{L}$, then $p(\ell) = \Phi(c)[^{id}/\underline{\texttt{\$clientId}}]$ for some c and id such that $c \in k(dev(\ell))$ and $\Phi(c)[^{id}/\underline{\texttt{\$clientId}}] \models (\texttt{Con}, id)$. We assume that $\ell' \in \mathbb{L}'$, and show by cases on the rules in Figure 4 that the same properties hold for σ', namely $p'(\ell') = \Phi(c')[^{id'}/\underline{\texttt{\$clientId}}]$ for some c' and id' such that $c' \in k(dev(\ell'))$ and $\Phi(c')[^{id'}/\underline{\texttt{\$clientId}}] \models (\texttt{Con}, id')$.

Case (CON): for $\ell' \in \mathbb{L}'$ to be true it must be that either $\ell' \in \mathbb{L}$, or ℓ' is the connection in the message m. In the former case, the thesis follows by induction hypothesis since $p \triangleleft [\ell' \mapsto P](\ell) = p(\ell)$. In the latter case, the thesis holds thank to the premises of the derivation rule, which coincide with our desiderata.

Cases (SUB) and (UNSUB): notice that $\mathbb{L}' = \mathbb{L}$ and $p' = p$, thus for $\ell' \in \mathbb{L}'$ to be true it must be that $\ell' \in \mathbb{L}$ and we can apply the induction hypothesis.

Case (DISC): induction hypothesis suffices because it must be that $\ell' \in \mathbb{L}$, and it holds by construction that $p'(\ell'') = p(\ell'')$ for each ℓ'' for which both functions are defined.

Default idel move: if not rule applies then $\sigma = \sigma'$ and the result trivially follows from induction hypothesis.

□

Lemma 2. *Let σ be a broker state such that $\sigma_\perp \rightarrow^* \sigma = (\mathbb{L}, p, \varsigma)$, if $p(\ell)$ is defined, then $\ell \in \mathbb{L}$.*

Proof. We prove the property by induction on the length of the execution π from $\sigma_\perp$ to σ. The state $\sigma_\perp$ vacuously satisfies the lemma, as the domain of p is empty. Assume $\pi = \pi' \odot (\sigma \xrightarrow{m} \sigma')$ where $\sigma = (\mathbb{L}, p, \varsigma)$ and $\sigma' = (\mathbb{L}', p', \varsigma')$, and consider the rules in Figure 4. For the (CON) rule, $p'(\ell')$ is defined only if either $p(\ell')$ is defined, or if ℓ' is the connection in the m message. In the former case, the thesis follows by induction hypothesis; in the latter, it holds by construction. If the rule (DISC) is used to build the transition, then $p'(\ell')$ defined implies that also $p(\ell')'$ is also defined, and $\ell' \neq \ell''$ with ℓ'' the connection in m. Then the hypothesis follows by induction hypothesis. All the other cases are trivially satisfied by induction hypothesis, because $p' = p$ and $\mathbb{L}' = \mathbb{L}$. □

Lemma 3. *Let σ be a broker state such that $\sigma_\perp \rightarrow^* \sigma = (\mathbb{L}, p, \varsigma)$, and let $\ell \in \varsigma(tf)$, then $p(\ell) = \Phi(c)[^{id}/\underline{\texttt{\$clientId}}]$ for some c and id such that $\Phi(c)[^{id}/\underline{\texttt{\$clientId}}] \models (\texttt{Sub}, tf)$, $c \in k(dev(\ell))$ and $\Phi(c)[^{id}/\underline{\texttt{\$clientId}}] \models (\texttt{Con}, id)$.*

Proof. We prove the property by induction on the length of the execution π from $\sigma_\perp$ to σ. The state $\sigma_\perp$ vacuously satisfies the lemma. Assume $\pi = \pi' \odot (\sigma \xrightarrow{m} \sigma')$ where $\sigma = (\mathbb{L}, p, \varsigma)$ and $\sigma' = (\mathbb{L}', p', \varsigma')$. Then, by induction hypothesis, if $\ell \in \varsigma(tf)$, then $p(\ell) = \Phi(c)[^{id}/\underline{\texttt{\$clientId}}]$ for some c and id such that $\Phi(c)[^{id}/\underline{\texttt{\$clientId}}] \models (\texttt{Sub}, tf)$, $c \in k(dev(\ell))$ and $\Phi(c)[^{id}/\underline{\texttt{\$clientId}}] \models (\texttt{Con}, id)$.

We assume that $\ell' \in \varsigma'(tf')$, and show by cases on the rules in Figure 4 that the same properties hold for σ', namely $p'(\ell') = \Phi(c')[^{id'}/\underline{\texttt{\$clientId}}]$ for some c' and id' such that $\Phi(c')[^{id'}/\underline{\texttt{\$clientId}}] \models (\texttt{Sub}, tf')$, $c' \in k(dev(\ell'))$ and $\Phi(c')[^{id'}/\underline{\texttt{\$clientId}}] \models (\texttt{Con}, id')$.

Case (CON): the result follows from the fact that $\varsigma = \varsigma'$, thus $\ell' \in \varsigma'(tf')$ implies $\ell' \in \varsigma(tf)$. In addition, ℓ'' of m is fresh in $\mathbb{L}$, thus $p(\ell') = p' \triangleleft [\ell' \mapsto P](\ell') = p(\ell')$.

Case (SUB): if the topicfilter tf'' in m is tf', then from Lemma 1 it holds that $\ell' \in \mathbb{L}$ implies $p(\ell') = p'(\ell') = \Phi(c')[^{id'}/\underline{\texttt{\$clientId}}]$ for some c' and id' such that $c' \in k(dev(\ell'))$ and $\Phi(c')[^{id'}/\underline{\texttt{\$clientId}}] \models (\texttt{Con}, id')$. Finally, the last condition of the derivation rule guarantees that $p'(\ell') \models (\texttt{Sub}, tf')$.
If tf'' in m is not tf' instead, it must be that tf' is some tf such that $\ell' \in \varsigma(tf)$ an the result follows by induction hypothesis.

Cases (UNSUB), (DISC), and default idel move Induction hypothesis suffices because it must be that $\ell' \in \varsigma(tf')$, and it holds by construction that $p'(\ell'') = p(\ell'')$ for each ℓ'' for which both functions are defined.

□

We now prove correctness and completeness of the information flow graph separately.

Lemma 4. *Given a configuration I, two devices $d, d' \in \mathbb{D}$ and a topic t, if a feasible execution π exists such that $d \overset{t}{\Rightarrow} d' \in Tr(\pi)$, then two certificates $c, c' \in \mathbb{C}$ also exist with $c \in k(d)$ and $c' \in k(d')$, and such that the information flow graph of I contains both (c, t) and (t, c').*

Proof. Take d, d', t and π, such that $d \overset{t}{\Rightarrow} d' \in Tr(\pi)$. We prove the desideratum by induction over the conditions of Definition 11.

As base case, assume that $\pi = \sigma \xrightarrow{m} \sigma'$, with $\sigma = (\mathbb{L}, p, \varsigma)$, $m = (\ell, \texttt{Pub}(t))$ and $d' \in Rec(\sigma, t)$, and $p(\ell) \models (\texttt{Pub}, t)$. By Lemma 2, $\ell \in \mathbb{L}$, and thus it holds by Lemma 1 that $p(\ell) = \Phi(c)[^{id}/\underline{\texttt{\$clientId}}]$ for some c and id such that $c \in k(dev(\ell))$ and $\Phi(c)[^{id}/\underline{\texttt{\$clientId}}] \models (\texttt{Con}, id)$. Therefore, we know that $t \in \mathbb{T}^O_c$ by Definition 12, and thus, by Definition 13, (c, t) is an arc of the information flow graph.

By Definition 10, $d' \in Rec(\sigma, t)$ implies t is in $\mathcal{L}(tf^{\texttt{mqtt}})$ for some tf such that $\ell \in \varsigma(tf)$ and $p(\ell) \models (\texttt{Rec}, t)$ for some ℓ such that $dev(\ell) = d'$. Since $\ell \in \varsigma(tf)$, by Lemma 3, a certificate c' must exists such that $p(\ell) = \Phi(c')[^{id'}/\underline{\texttt{\$clientId}}]$ and $\Phi(c')[^{id'}/\underline{\texttt{\$clientId}}] \models (\texttt{Sub}, tf)$ for some id' such that $c' \in k(dev(\ell'))$ and $\Phi(c')[^{id'}/\underline{\texttt{\$clientId}}] \models (\texttt{Con}, id')$. Therefore, we know that $t \in \mathbb{T}^I_{c'}$ by Definition 12, and thus, by Definition 13, (t, c') is an arc of the information flow graph.

Finally, we consider the induction step and assume $\pi = \sigma \xrightarrow{m} \sigma' \odot \pi'$. Then either $d \overset{t}{\Rightarrow} d' \in Tr(\sigma \xrightarrow{m} \sigma')$ or $d \overset{t}{\Rightarrow} d' \in Tr(\pi')$, and in both cases we can directly rely on the induction hypothesis. □

Lemma 5. *Given a configuration and its information flow graph G, two certificates $c, c' \in \mathbb{C}$, and a topic t, if both (c, t) and (t, c') are arcs in G, then for all devices $d, d' \in \mathbb{D}$ such that $c \in k(d)$ and $c' \in k(d')$, a feasible execution π exists satisfying $d \overset{t}{\Rightarrow} d' \in Tr(\pi)$.*

Proof. Take c, c', d, d' and t such that $c \in k(d)$, $c' \in k(d')$, and assume both (c, t) and (t, c') are arcs of the information flow graph. By Definition 13, $t \in \mathbb{T}^O_c$ and $t \in \mathbb{T}^I_{c'}$; thus we know that id, id' and tf exists such that we can build the following feasible execution π in accordance with the rules of Figure 4, where $dev(\ell) = d$, $dev(\ell') = d'$, and $\Phi_{c,id}$ stands

for $\Phi(c)[\,^{id}/_{\underline{\texttt{\$clientId}}}]$.

$$\begin{aligned}
&\sigma_\perp \xrightarrow{(\ell,\texttt{Con}(c,id))} \\
&\quad (\{\ell\}, \{\ell \mapsto \Phi_{c,id}\}, \varsigma_\perp) \\
&\quad \xrightarrow{(\ell',\texttt{Con}(c',id'))} \\
&\quad (\{\ell,\ell'\}, \{\ell \mapsto \Phi_{c,id}, \ell' \mapsto \Phi_{c',id'}\}, \varsigma_\perp) \\
&\quad \xrightarrow{(\ell',\texttt{Sub}(tf))} \\
&\quad (\{\ell,\ell'\}, \{\ell \mapsto \Phi_{c,id}, \ell' \mapsto \Phi_{c',id'}\}, \{tf \mapsto \{\ell'\}\}) \\
&\quad \xrightarrow{(\ell,\texttt{Pub}(t))} \\
&\quad (\{\ell,\ell'\}, \{\ell \mapsto \Phi_{c,id}, \ell' \mapsto \Phi_{c',id'}\}, \{tf \mapsto \{\ell'\}\})
\end{aligned}$$

The proof concludes by showing that $d \overset{t}{\Rightarrow} d' \in Tr(\pi)$. In particular, it holds by Definition 11 that the conditions of $t \in \mathbb{T}_c^O$ and $t \in \mathbb{T}_{c'}^I$ imply $d \overset{t}{\Rightarrow} d' \in Tr(\sigma \xrightarrow{(\ell,\texttt{Pub}(t))} \sigma)$, with σ the last broker state of π. □

Theorem 1. *Given a configuration and its information flow graph G, it holds that:*

1) *if G contains both the arcs (c,t) and (t,c') then for each pair of devices d, d' such that $c \in k(d)$ and $c' \in k(d')$ a feasible execution π* exists *with $d \overset{t}{\Rightarrow} d' \in Tr(\pi)$; and*
2) *if d, d' exists such that $c \in k(d)$ and $c' \in k(d')$, and there is a feasible execution π such that $d \overset{t}{\Rightarrow} d' \in Tr(\pi)$, then G contains both the arcs (c,t) and (t,c').*

Proof. By Lemma 4 and 5. □

We address now reachability over the information flow graph, and its correspondence to multi-steps communication traces. A useful property of feasible executions is that they can be composed, thanks to `Disc` which allows extending each feasible execution to one from $\sigma_\perp$ to $\sigma_\perp$.

Lemma 6. *Let π, π' be feasible executions of a configuration, then a feasible execution π'' exists such that $Tr(\pi'') = Tr(\pi) \odot Tr(\pi')$.*

Proof. Let π and π' be as follows:

$$\pi = \sigma_\perp \xrightarrow{m_1} \sigma_1 \xrightarrow{m_2} \dots \xrightarrow{m_n} \sigma_n$$
$$\pi' = \sigma_\perp \xrightarrow{m'_1} \sigma'_1 \xrightarrow{m'_2} \dots \xrightarrow{m'_m} \sigma'_m$$

Let also $\sigma_n = (\mathbb{L}, p, \varsigma)$, with $\mathbb{L} = \{\ell_i\}_{i=1}^k$ and let $m''_i = (\ell_i, \texttt{Disc})$ for all i from 1 to k. Then it holds by construction that the following π'' is a feasible execution:

$$\begin{aligned}
\pi'' = \sigma_\perp &\xrightarrow{m_1} \sigma_1 \xrightarrow{m_2} \dots \xrightarrow{m_n} \\
\sigma_n &\xrightarrow{m''_1} \sigma''_1 \xrightarrow{m''_2} \sigma''_2 \dots \xrightarrow{m''_k} \\
\sigma_\perp &\xrightarrow{m'_1} \sigma'_1 \xrightarrow{m'_2} \dots \xrightarrow{m'_m} \sigma'_m.
\end{aligned}$$

Moreover, by Definition 11, $Tr(\sigma_n \xrightarrow{m''_1} \dots \xrightarrow{m''_k}) = \emptyset$, and also $Tr(\pi'') = Tr(\pi) \odot \emptyset \odot Tr(\pi') = Tr(\pi) \odot Tr(\pi')$. □

Corollary 1. *Given a configuration and its information flow graph G, for each pair of certificates c and c' it holds that:*

1) *if G has an information flow from c to c' then for each d and d' with $c \in k(d)$ and $c' \in k(d')$, a feasible execution π* exists *such that a trace from d to d' is in $Tr(\pi)$; and*
2) *if d, d' and π feasible exist such that $c \in k(d)$, $c' \in k(d')$, and $Tr(\pi)$ contains a trace from d to d', then G has an information flow from c to c'.*

Proof. The result follows by definition from Theorem 1 and Lemma 6. □

We establish the following consistency result between the intentional description of the policy interrogation of Definition 4 and the predicate of Definition 14.

Lemma 7. *Given a configuration I, the permission predicate $\psi_{(c,id,\alpha,\rho)}$ is true if and only if $\Phi(c)[\,^{id}/_{\underline{\texttt{\$clientId}}}] \models (\alpha, \rho)$.*

Proof. The result trivially holds by definition, and can be proved by induction on the cardinality of $\Phi(c)$. □

The following lemma ensures that two certificates are reachable in one step in the information flow graph if and only if the corresponding first order predicate is true.

Lemma 8. *Given a configuration and its information flow graph $G = (N, E)$, then $(c,t) \in E$ and $(t,c') \in E$ if and only if $F_{c,c'}$ of Definition 16 is true when instantiating the topic with t.*

Proof. We first note that, by Definition 13, $(c,t) \in E$ and $(t,c') \in E$ hold if and only if there exists a t such that $t \in \mathbb{T}_c^O \cap \mathbb{T}_{c'}^I$. Notice that Lemma 7 implies:

$$t \in \mathbb{T}_c^O \text{ iff } \mathcal{O}_c(t) \qquad \text{and} \qquad t \in \mathbb{T}_{c'}^I \text{ iff } \mathcal{I}_{c'}(t).$$

Then, by Definition 16, this is equivalent to t satisfying $F_{c,c'}$. □

The following lemma ensures that each pair of arcs in the information flow graph that share the topic has a corresponding topic node in the symbolic information flow graph and vice versa.

Theorem 2. *Given a configuration, there is an information flow from c to c' if and only if there is a symbolic information flow from c to c'.*

Proof. The statement follows by repeated iteration of Lemma 8 and by noticing that according to Definition 16, $F_{c,c'}$ is satisfied if and only if $(c, F_{c,c'})$ and $(F_{c,c'}, c')$ are arcs of the symbolic information flow graph. □

Theorem 3 (Complexity)**.** *Let $(\mathbb{C}, \mathbb{P}, \varphi, \mathbb{D}, k)$ be a configuration, and let $f(n)$ be the cost for deciding the satisfiability of quantifier-free predicates of size n using regular expressions and string theories. Then, the time required by Algorithm 1 is at most $O(|\mathbb{C}|^2 \cdot (|\overline{P}| \cdot |\overline{\mathcal{R}}| + f(|\overline{P}| \cdot |\overline{\mathcal{R}}|)))$, where $|\overline{P}|$ and $|\overline{\mathcal{R}}|$ are the size of the largest policy and of the largest resource expression.*

Proof. Algorithm 1 iterates over each pair of certificates (c, c') and for each pairs it builds the predicate $F_{c,c'}$ and checks its satisfiability. The overall cost is thus $O(|\mathbb{C}|^2 \cdot B \cdot S)$, where B and S are the cost of building and checking satisfiability of $F_{c,c'}$, respectively.

To estimate the value of B, consider a request r with certificate c. From Definition 14 we have that the formula ψ_r is of length at most $O(\sum_{s=(\varepsilon,\alpha,\mathcal{R})\in\Phi(c)} |\mathcal{R}|)$. We can approximate the expression above as $O(\overline{P} \cdot \overline{\mathcal{R}})$ by considering the worst case, i.e. the policy containing the greatest number of statements $\overline{P}$ and the statement with the greatest number of resource expressions $\overline{\mathcal{R}}$. Although each $F_{c,c'}$ is the combination of five expressions built though $\psi\_$, its length is still at most $O(\overline{P} \cdot \overline{\mathcal{R}})$ (see Definition 15 and Definition 16). The cost of building $F_{c,c'}$ is thus $B = O(\overline{P} \cdot \overline{\mathcal{R}})$.

To estimate the value of S, consider that checking the satisfiability of $F_{c,c'}$ through Algorithm 2 depends on the decision procedure used by the solver, which is invoked a fixed number of times (ten times in the worst case). Overall, the worst case complexity is $O(|\mathbb{C}|^2 \cdot (\overline{P} \cdot \overline{\mathcal{R}} + f(\overline{P} \cdot \overline{\mathcal{R}})))$, where f is the cost of the algorithm used by the chosen SMT solver for dealing with regular expressions and string concatenation. □

APPENDIX B
COMPARING DIFFERENT SMT SOLVERS

Table III compares the scalability of IOT:POKER using cvc5 and Z3 as the SMT solver on the experiments described in Section VI. The experiments of Table III were carried out by generating a set of random devices from which we derive the configuration on which we tested the two versions of the tool. The solutions found by using the two solvers coincide as expected, so do the average number of nodes and the times the hard strategy is required. Our results show that cvc5 performs better than Z3 almost all the time.

Table III
IoT:Poker Scalability on real-world configurations: comparison between cvc5 and Z3 as backend SMT solver.

devices	nodes avg.	hard strategies avg.	building time with (s) min.		avg.		max.		query time (s) avg.
			cvc5	**Z3**	**cvc5**	**Z3**	**cvc5**	**Z3**	
20	264.6	1.8	0.42	1.22	2.87	4.94	12.00	10.69	0.0060
40	973.0	4.5	2.30	6.17	6.28	12.64	16.55	20.28	0.0086
60	2211.2	5.6	2.93	7.04	9.22	18.22	23.39	30.00	0.0108
80	3903.9	13.6	5.93	18.67	16.34	30.72	29.08	47.88	0.0136
100	6098.1	16.7	7.50	18.89	19.46	39.82	36.55	58.76	0.0167
120	8764.2	21.5	13.58	35.17	24.69	52.35	41.49	87.98	0.0198
140	12080.1	28.7	18.61	42.75	31.06	64.81	46.63	100.06	0.0244
160	15415.9	29.6	20.03	51.89	33.01	72.11	44.43	90.16	0.0280
180	19571.7	37.0	26.61	69.91	37.87	86.89	52.79	109.81	0.0318
200	24209.9	45.8	31.62	78.76	46.06	103.63	57.34	125.65	0.0349
220	29066.2	50.5	32.39	95.42	50.93	117.21	62.02	129.81	0.0393
240	34732.6	59.0	42.11	114.48	57.68	130.17	62.41	141.43	0.0430
258	40019.0	65.0	62.35	141.24	62.67	144.20	63.53	152.19	0.0473