Access Control Model for AWS Internet of Things

Smriti Bhatt(✉), Farhan Patwa, and Ravi Sandhu

Department of Computer Science and Institute for Cyber Security,
University of Texas at San Antonio, One UTSA Circle, San Antonio, TX 78249, USA
bhattsmriti1@gmail.com, {farhan.patwa,ravi.sandhu}@utsa.edu

Abstract. Internet of Things (IoT) has received considerable attention in both industry and academia in recent years. There has been significant research on access control models for IoT in academia, while industrial deployment of several cloud-enabled IoT platforms have already been introduced. However, as yet there is no consensus on a formal access control model for cloud-enabled IoT. Currently, most of the cloud-enabled IoT platforms utilize some customized form of Role-Based Access Control (RBAC), but RBAC by itself is insufficient to address the dynamic requirements of IoT. In this paper, we study one of the commercial cloud-IoT platform, AWS IoT, and develop a formal access control model for it, which we call AWS-IoTAC. We do this by extending AWS cloud's formal access control (AWSAC) model, previously published in the academic literature, to incorporate the IoT specific components. The AWS-IoTAC model is abstracted from AWS IoT documentation and has been formalized based on AWSAC definitions. We show how this model maps to a recently proposed Access Control Oriented (ACO) architecture for cloud-enabled IoT. We demonstrate a smart-home use case in AWS IoT platform, and inspired by this use case, we propose some Attribute-Based Access Control (ABAC) extensions to the AWS-IoTAC model for enhancing the flexibility of access control in IoT.

Keywords: Internet of Things · Devices · Virtual objects · Attributes · Attribute-based access control

1 Introduction

Security is an essential requirement for the Internet of Things (IoT), especially as deployments grow. The number of connected devices is increasing exponentially. According to Gartner, there will be more than 20 billion connected devices by 2020 [5]. This has given rise to an attractive and new attack surface. Access control is an essential component of security solutions for IoT. Accordingly, several access control models for IoT have been proposed. Ouaddah et al. [25] provide a recent survey of these. Meanwhile, dominant cloud providers, such as Amazon Web Services (AWS) [1], Microsoft Azure [7], and Google Cloud Platform (GCP) [6], have built upon their existing cloud services and resources

Z. Yan et al. (Eds.): NSS 2017, LNCS 10394, pp. 721–736, 2017.
DOI: 10.1007/978-3-319-64701-2_57

to launch IoT services. Azure and GCP utilize some customized form of role-based access control (RBAC) [15,27] with predefined roles and groups for their access control requirements in the cloud. GCP uses RBAC for its IoT solutions authorization [9]. AWS uses a policy-based access control mechanism for its cloud and IoT services [1,2]. Unlike Azure cloud, Azure IoT has adopted policy-based access control to specify IoT authorizations [4]. However, a formal access control model for real-world cloud-enabled IoT platforms is still lacking.

In this paper, we study and investigate AWS and its IoT service, leading to a formal access control model called AWS-IoTAC. This model is abstracted from dispersed AWS IoT documentation available, along with our exercises on this service to validate our understanding of the IoT functionality. AWS-IoTAC builds upon the AWS Access Control (AWSAC) model developed by Zhang et al. [29], for AWS access control in general.

The IoT services require new concepts beyond basic access control in the cloud. While developing an access control model for IoT, conceptualizing the model in context of a well-defined IoT architecture is useful. A layered access control oriented (ACO) architecture for cloud-enabled IoT has been proposed by Alshehri and Sandhu [12]. We map different entities of our model with the four layers of ACO architecture to underscore the relevance of our model with a cloud-enabled IoT architecture especially designed from an access control perspective. We also demonstrate and configure a smart-home use case in the AWS IoT platform which depicts the applicability of our model in addressing IoT authorizations in AWS.

With billions of connected devices in the near future, it will become inevitable for IoT to adopt a flexible access control model, such as attribute-based access control (ABAC) [18,19], for meeting dynamic access control requirements of the IoT services. In ABAC, attributes (properties), represented as name-value pairs, of users and resources are utilized to determine user accesses on resources. AWS IoT supports a partial form of ABAC with attributes for the IoT devices, however, the use of these attributes in access control policies is limited. Therefore, we propose ABAC enhancements to our (AWS-IoTAC) model for incorporating a complete form of ABAC in it.

A summary of our contributions is given below.

- We develop a formal access control model for AWS IoT, which we call AWS-IoTAC.
- We present a smart-home IoT use case which clearly shows how our model addresses the authorizations in a cloud-enabled IoT platform.
- We propose ABAC enhancements for AWS-IoTAC to include more flexible and fine-grained access policies.

The rest of the paper is organized as follows. Section 2 discusses related work, including the AWSAC model. AWS-IoTAC model is presented and defined in Sect. 3. A smart-home use case that utilizes the AWS-IoTAC model is demonstrated in Sect. 4. Section 5 proposes some extensions to AWS-IoTAC. Finally, we conclude the paper with possible future directions in Sect. 6.

2 Related Work and Background

2.1 Related Work

There has been significant research in IoT access control models, as recently surveyed by Ouaddah et al. [25]. Many of these models are based on capability-based access control (CAPBAC) [17], role-based access control (RBAC) [15,27], while there are a few utilizing attribute-based access control (ABAC) [18,19]. In [16], a centralized CAPBAC model has been proposed based on a centralized Policy Decision Point (PDP). Whereas, a fully decentralized CAPBAC model for IoT is presented in [17]. However, a fully centralized or a fully decentralized approach may not be appropriate for managing the access control needs in a dynamic IoT architecture. Mahalle et al. [23] proposed an identity establishment and capability-based access control scheme for authentication and access control in IoT. Besides CAPBAC, a RBAC model is used for IoT in [22] where a thing's accesses are determined based on its roles. Similarly, Zhang and Tian [28] proposed an extended role-based access control model for IoT where access is granted based on the context information collected from system and user environment. These RBAC models for IoT still suffer from RBAC's limitations, such as role-explosion [26]. A hybrid access control model (ARBHAC) based on RBAC and ABAC is proposed by Sun et al. [20] to handle large number of dynamic users in IoT. Here attributes are used to make user-role assignments, and then a user's roles determine access on resources or things. This approach is similar to *dynamic roles* [11,21], where roles are dynamically assigned to users based on their attributes. However, ARBHAC lacks utilization of user, thing, environment and application attributes available in more general ABAC models.

Our model significantly differs from the existing models discussed above, especially in its nature of being an access control model developed for a real-world cloud-enabled IoT platform that is managed by the largest cloud service provider, Amazon Web Services (AWS) [1]. Another distinguishing feature of our work is to identify the applicability of user attributes and attributes of IoT things (things/devices requesting access to other things/devices, and things/devices on which the access is being requested) in IoT access control. We strongly believe that ABAC models are the best approach to address access control requirements of the rapidly evolving IoT arena.

2.2 AWS Access Control Model (AWSAC)

An access control model for AWS cloud services was developed by Zhang et al. [29]. We briefly describe the AWS Access Control (AWSAC) model and its formal definitions here, which in turn forms a base for the AWS-IoTAC model presented in the next section. The AWSAC model within a single AWS account is shown in Fig. 1, with formal definitions presented in Table 1. AWSAC has seven components: *Accounts (A), Users (U), Groups (G), Roles (R), Services (S), Object Types (OT),* and *Operations (OP).* **Accounts** are basic resource containers in AWS, which allows customers to own specific cloud resources, and

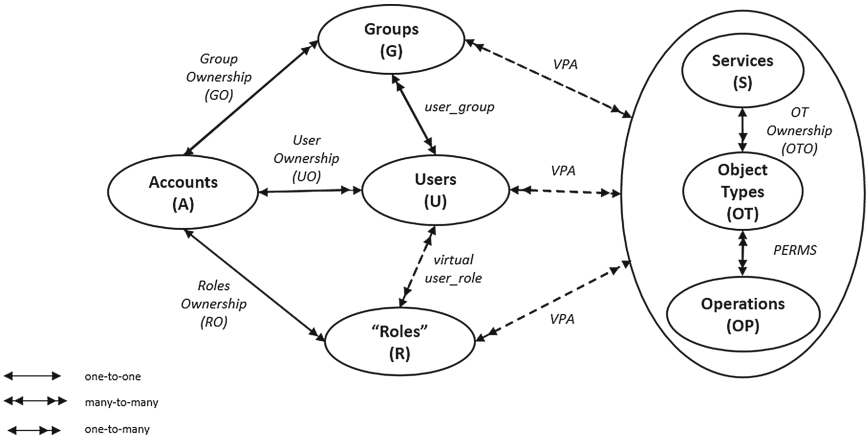


Fig. 1. AWS access control within a single account [29]

Table 1. AWSAC model components [29]

Definition 1
- A, U, G, R, S, OT and OP are finite sets of accounts, users, groups, roles, services, object types, and operations respectively
- User Ownership $(UO) : U \rightarrow A$, is a function mapping a user to its owning account, equivalently a many-to-one relation $UO \subseteq U \times A$
- Group Ownership $(GO) : G \rightarrow A$, is a function mapping a group to its owning account, equivalently a many-to-one relation $GO \subseteq G \times A$
- Role Ownership $(RO) : R \rightarrow A$, is a function mapping a role to its owning account, equivalently a many-to-one relation $RO \subseteq R \times A$
- Object Type Ownership$(OTO) : OT \rightarrow S$, is a function mapping an object type to its owning service, equivalently a many-to-one relation $OTO \subseteq OT \times S$
- $PERMS = OT \times OP$, is the set of permissions
- Virtual Permission Assignment (VPA): $VPA \subseteq (U \cup G \cup R) \times PERMS$, is a many-to-many virtual relation resulting from policies attached to users, groups, roles and resources
- $user\_group \subseteq U \times G$ is a many-to-many mapping between users and groups where users and groups are owned by the same account
- virtual user_role $(VUR) : VUR \subseteq U \times R$ is a virtual relation resulting from policies attached to various entities (users, roles, groups), where users use AssumeRole action to acquire/activate a role authorized in VUR

serve as the basic unit of resource usage and billing. **Users** represent individuals who can be authenticated by AWS and authorized to access cloud resources through an account. A user who owns an account can create other users inside that account and can assign them specific permissions on resources, and thus is an administrator. **Groups** are a set of user groups. The **user_group** relation specifies the user to group assignment. **"Roles"** in AWS, unlike standard RBAC roles, are used for establishing trust relationships between users and resources in different AWS accounts. Users can be assigned roles through the *AssumeRole* action, and permissions assigned to these roles allows these users gain access to corresponding cloud resources. The user-role mapping is specified through **virtual user_role** relation. To distinguish the AWS "roles" from RBAC roles, quotation marks are used in Fig. 1. For simplicity, we understand roles to signify "roles" in rest of the paper, unless otherwise specified. **Services** refer to AWS cloud services. **Object Types** represents a specific type of an object in a particular cloud service, such as virtual machines. **Operations** represent allowed operations on the object types based on an access control policy attached to them or their owning services.

AWS utilizes a policy-based access control mechanism. An AWS **policy** is a JSON file which includes permissions defined on services and resources in the cloud. It comprise of three main parts (or tags) *Effect, Action* and *Resources*, and optional *Conditions*. A policy can be attached to a user, a group, a role or a specific cloud resource. **Virtual Permission Assignment** is the process of virtually assigning permissions to users, roles, and groups through attaching policies to these entities. In cases where a policy is attached to a resource, a specific *Principal* (an account, a user or a role) needs to be specified in the policy. There could be multiple permissions defined in one policy, and multiple policies can be attached to one entity.

3 Access Control in AWS Internet of Things

3.1 AWS IoT Access Control (AWS-IoTAC) Model

AWS IoT is an IoT platform managed by one of the leading cloud service provider, Amazon Web Services (AWS). It allows secure communication between connected IoT devices and applications in the AWS cloud [2]. An access control model for AWS IoT, a cloud-enabled IoT platform, involves different entities in the IoT space, and should define how these entities are authorized to interact with each other securely. We incorporate the entities involved in access control and authorization in the AWS IoT service into the AWSAC model so as to develop the AWS-IoTAC model. AWS-IoTAC is based on meticulous exploration of the extensive documentation on AWS IoT and our hands-on experiments on this service to verify our understanding.

The AWS-IoTAC model is shown in Fig. 2 along with its different components. Since it is developed on top of the AWSAC model, it consists of all the components and relations of AWSAC with additional set of components and

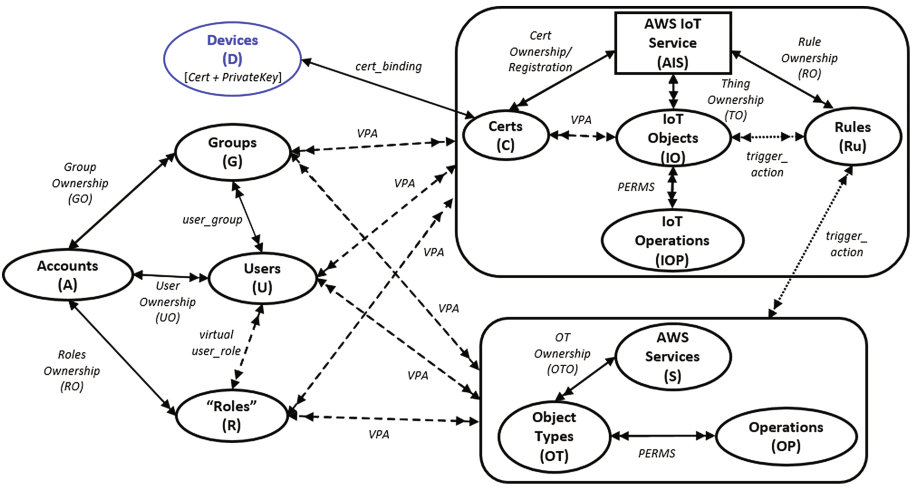


Fig. 2. AWS IoT access control (AWS-IoTAC) model within a single account

Table 2. AWS-IoTAC model – additional components and relations

Definition 2
- *AWS IoT Service (AIS) is one of the Services(S) in AWS*
- *C, D, IO, IOP, and Ru are finite sets of X.509 certificates, physical IoT devices, IoT objects, IoT operations, and rules defined in the rules engine of AIS respectively*
- *Cert Ownership/Registration $(CO) : C \rightarrow AIS$, is a function mapping a certificate to its owning service (AIS), equivalently a many-to-one relation $CO \subseteq C \times AIS$*
- *Rules Ownership $(RO) : Ru \rightarrow AIS$, is a function mapping a rule to its owning service (AIS), equivalently a many-to-one relation $RO \subseteq Ru \times AIS$*
- *Thing Ownership $(TO) : IO \rightarrow AIS$, is a function mapping the IoT objects to its owning service (AIS), equivalently a many-to-one relation $TO \subseteq IO \times AIS$*
- *$PERMS = OT \times OP$, is the set of permissions (including IoT permissions)*
- *Virtual Permission Assignment (VPA): $VPA \subseteq (U \cup G \cup R \cup C) \times PERMS$, is a many-to-many virtual relation resulting from policies attached to users, groups, roles, certificates, and resources*
- *cert_binding $\subseteq C \times D$ is a mutable one-to-one relation between X.509 certificate and IoT devices within a single account*
- *trigger_action $\subseteq Ru \times (IO \times S)$ represents a many-to-many mapping between rules and IoT objects and AWS services on which a rule triggers action(s)*

relations associated with the AWS IoT service. The additional or modified components and relations are formally defined in Table 2, and informally discussed below. There are six additional components in the AWS-IoTAC model. **AWS IoT Service (AIS)** is the new IoT service in AWS. It owns different entities to support IoT devices and their underlying authorization in the cloud. We represent AIS as a separate entity in our model to emphasize its importance and clearly show other components and relations associated with it. The rectangular box of AIS emphasize its singleton existence in AWS. **Certs (C)** is a set of X.509 certificates [10], issued by a trusted entity, the certificate authority (CA). AIS can generate X.509 certificates for the IoT clients, or allow the use of certificates created by the clients as long as they are signed by a registered CA in the AWS IoT service. Certs are used by MQTT based clients (IoT devices, applications) to authenticate to AIS. MQTT, an OASIS standard, is a machine-to-machine (M2M) lightweight publish/subscribe messaging protocol, especially designed for constrained devices [8]. **Devices (D)** represent a set of connected IoT devices, such as sensors, light bulbs. The devices can exist independent of AIS, thus, we show them in a different color in the model. A valid X.509 certificate and its private key need to be copied onto the device, along with a root AWS CA certificate before authentication and establishment of a secure communication channel with the AWS IoT service. The certificates to devices association is done through the **cert_binding** relation. In the AWS IoT platform, one certificate can be attached to many things/devices. Similarly, many certificates can be copied onto one IoT device. However, in our model, we assume *cert_binding* is an one-to-one association between devices and certificates for better authorization management, and is mutable in nature so can be changed by an administrator in cases of certificate expiry or revocation. In AWS IoT, the access control policies are attached to certificates, and are enforced on physical IoT devices associated with these certificates.

IoT Objects (IO) represent virtual IoT objects in the cloud. *Virtual objects* are the digital counterparts of real physical devices, or standalone logical entities (applications) in the virtual space [24]. In AWS IoT, a *Thing* and a *Thing Shadow* represent the IoT objects which are the virtual counterparts of real physical IoT devices in the cloud. For each IoT device, we assume that there is at least one *thing* with its *thing shadow* instantiated in the cloud, which provides a set of predefined MQTT topics/channels (associated with this device) to allow interaction with other IoT devices and applications, even when the device is offline. *Thing shadow* maintains the identity and last known state of the associated IoT device. **IoT Operations (IOP)** are a set of operational operations defined for IoT service, and do not include the administrative operations, such as create things, certificates, etc. The basic set of IoT operations can be categorized based on the communication protocols used by IoT devices and applications to communicate with the AWS IoT service. For MQTT clients, four basic IoT operations are available: *iot:Publish* allows devices to publish a message to a MQTT topic, *iot:Subscribe* allows a device to subscribe to a desired MQTT topic, *iot:Connect* allows a MQTT client to connect to the AWS IoT service, and

iot:Receive allows devices to receive messages from subscribed topics. Similarly, for HTTP clients, *iot:GetThingShadow* allows to get the current state of a thing shadow, *iot:UpdateThingShadow* allows to send messages to update/change the state of a thing shadow, and *iot:DeleteThingShadow* deletes a thing shadow. Whenever a device or application sends message to a virtual thing in the cloud, a new thing shadow is automatically created, if one does not already exist.

Rules (Ru) are simple SQL statements which trigger predefined actions based on the condition defined in the rule. A rule receives data from a device/thing and triggers one or more actions. The actions route the data from one IoT device to other IoT devices, or to other AWS services. Each rule must be associated with an IAM (Identity and Access Management) role which grants it permissions to access IoT objects and AWS services on which actions are triggered. The relation **trigger_action** represents a many-to-many mapping between rules and IoT objects and AWS services on which the rule triggers action(s). The access control policies in AWS have been modified to include IoT operations and resources, and are thereby named as IoT policies. AWS IoT utilizes both IoT policies and IAM policies to assign specific permissions to IoT devices, IAM users, and IoT applications. Consequently, **Virtual Permission Assignment (VPA)** has been updated to include the IoT policies, and these policies are attached to X.509 certificates. The policy attached to a certificate is enforced on the device which uses that certificate to connect and authenticate to the AWS IoT service. One policy can be attached to multiple certificates, or multiple policies can be attached to one certificate.

All the components and relations of our model are defined within the scope of a single AWS account. Cross-account authorizations are outside the scope of this paper. The components and relations of our model are based on current capabilities of the AWS IoT service. Although, there are many other components and relations associated with the AWS IoT service, we encompassed the most important ones from an access control perspective in our model.

3.2 ACO IoT Mapping

Here, we show relevance of the AWS-IoTAC model to the access control oriented (ACO) IoT architecture presented by Alshehri and Sandhu [12]. A mapping of different entities of our model with the ACO architecture is depicted in Fig. 3. Different entities map to different layers of ACO IoT architecture. Physical devices or things exist at *Object layer*, and virtual IoT things or resources maps to the *Virtual Object layer*. All the AWS cloud services and resources are at *Cloud Service layer*, and users and applications interacting with cloud and IoT devices exists at *Application layer*. The authorization policies are defined in the cloud. These policies enforce access control decisions for physical devices and applications (used by users) trying to access cloud and virtual IoT resources. AWS-IoTAC is generally compatible with the ACO architecture.

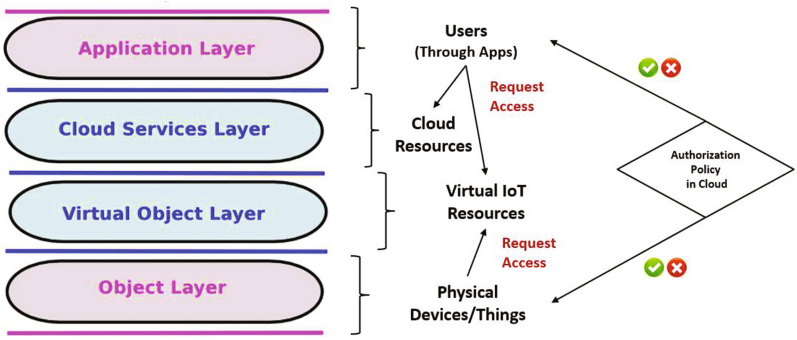


Fig. 3. AWS-IoTAC entities mapping to ACO architecture for cloud-enabled IoT

4 Use Case

In this section, we present a smart-home use case where a thermostat and two light bulbs are controlled through the AWS IoT service based on sensor inputs. Here we focus on interactions between IoT devices through the cloud. (A more complex example would involve different users and applications also interacting with IoT devices.) We demonstrate how the access control and authorization between different components are configured based on the AWS-IoTAC model.

4.1 Use Case Setup and Configuration

Figure 4 shows different connected devices, virtual things/objects, and AWS Cloud and IoT Services involved in the use case. We first created an AWS account to setup the use case in the AWS IoT service. Using AWS IoT management console, we created one virtual object (*thing*) for each physical device—two sensors,

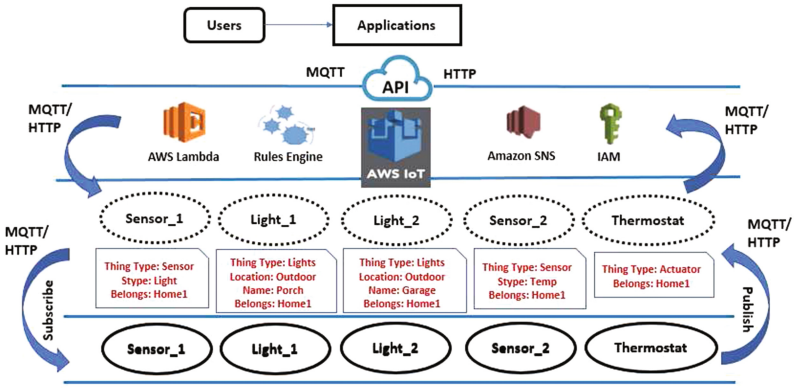


Fig. 4. Smart-home use case utilizing AWS IoT and cloud services

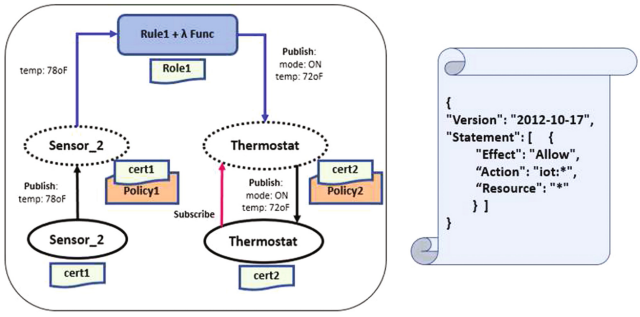
one thermostat and two light bulbs. A *thing* can have a *thing type* that stores configuration for similar things, and *thing attributes* (key-value pairs) representing properties of individual IoT devices. For example, *Sensor_1* has a *Sensor* thing type and has two attributes *SType* (sensor type) and *Belongs* (belongs to). The values for these attributes are set during thing creation. We also created X.509 certificates for each IoT thing/device using "one-click certificate creation" in the AWS IoT console. We then defined and attached appropriate authorization policies to the certificates. After policy attachment, appropriate certificate is attached to a virtual thing and is copied onto its corresponding physical device along with the private key of the certificate and an AWS root CA certificate. The CA certificate specifies the identity of the server, viz., AWS IoT server in this case. A device certificate is used during device authentication and specifies its authorization based on attached policies. We simulated the lights and thermostat devices using AWS SDKs (Node.js) [3] provided by AWS, and simulated sensors as MQTT clients using MQTT.fx tool [8]. All these devices use MQTT protocol to communicate to the AWS IoT service with TLS security.

Based on our use case scenarios, we utilized the *rules engine*, a part of the AWS IoT platform, to define rules and trigger desired actions. The actions include a *Lambda function* and notification to users by sending text messages through *Amazon Simple Notification Service (SNS)*. For each rule, an IAM "role" is associated with it to authorize the rule to access required AWS and AWS IoT services and resources.

4.2 Use Case Scenarios

Here, we discuss two scenarios of our use case and their relevant authorization aspects.

A. **Scenario 1:** This scenario involves a temperature sensor and a thermostat and is depicted in Fig. 5(a). A temperature sensor *Sensor_2* (shown in solid oval) senses the temperature and sends data to its thing shadow, *Sensor_2*



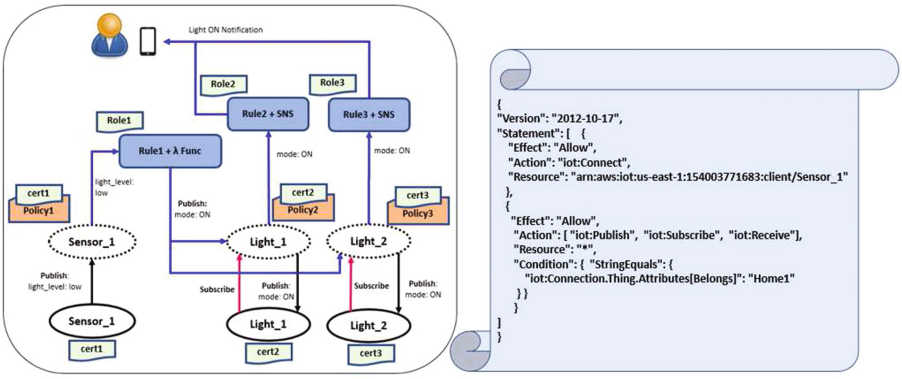
a. Use Case – Scenario 1

b. A Simple IoT Policy

Fig. 5. Smart-home use case scenario 1

(shown in dotted oval), in the AWS IoT platform. Based on *Sensor_2* data, a rule (Rule1) triggers a lambda function to change the state of the *Thermostat* by publishing an update message to its thing shadow (shown as the dotted oval). If the environment temperature is greater than 78 degree Fahrenheit, then the rule invokes a lambda function that publishes a message on *Thermostat* thing shadow to turn on the thermostat and set its temperature to 72 degree Fahrenheit. The physical thermostat (shown in solid oval) has subscribed to its shadow topics, hence, receives the update message and syncs its state with its thing shadow. For this scenario, we defined a simple authorization policy for both *Sensor_2* and *Thermostat*, as shown in Fig. 5(b). It allows an entity to do any IoT operation (e.g., publish, subscribe) on any resource in the AWS cloud. The policy is attached to the X.509 certificates which are copied onto the corresponding physical IoT devices (*Sensor_2* and *Thermostat*). In this example, the physical devices have full IoT access on all the resources in AWS IoT.

B. **Scenario 2:** A more comprehensive scenario with a fine-grained authorization policy is presented in Fig. 6. A light sensor, *Sensor_1*, monitors the light level of the environment and turns on outdoor lights, *Light_1* and *Light_2*, when the light level is low. When the lights are turned on, users (owner or resident) of the home get a text notification about the state change of the lights. For this scenario, we defined a more restrictive policy for *Sensor_1* where we utilized thing attributes in the *Condition* section of the policy. The policy is shown in Fig. 6(b), and comprises two policy statements—first to authorize a client to connect to AWS IoT only if its client ID is *Sensor_1*, and second to allow IoT publish, subscribe, and receive operations on all resources only if the client requesting access has a thing attribute *Belongs* with a value *Home1*. This policy employs thing attributes in making access control decisions. Thing attributes, as shown in Fig. 4, represent the characteristics of IoT things/devices.



a. Use Case – Scenario 2 b. A Fine-grained IoT Policy

Fig. 6. Smart-home use case scenario 2

Currently AWS IoT policy supports thing attributes of only those clients (devices/things) which are requesting access on resources in the AWS IoT service. A useful scenario would be to utilize the attributes of target resources on which IoT operations are performed. Suppose, a user wants *Sensor_1* to be able to publish data only on those lights which have an attribute *Location = Outdoor*. Currently, the AWS-IoTAC model cannot incorporate the attributes of target things/devices in IoT policies. This scenario, however, can be realized by means of rules and lambda functions as illustrated in the following. The code snippet of the lambda function is presented in Fig. 7. Here, we search for things that have an attribute key and value, *Location = Outdoor*, and get a list of such things, i.e., *Light_1* and *Light_2* in our use case. Once the list is obtained, a message (in JSON format) to turn on the lights is published to their shadow update topic, as shown in the Figure. The physical light bulbs receive the update message and change their states. As soon as the device state changes, a text message notification is sent to a user specified in the rules, *Rule_2* and *Rule_3*, through the AWS SNS service.

```
...
var params2 = {attributeName: 'Location',
               attributeValue: 'Outdoor'
};
iot.listThings(params2, function(err, data) {
  ...
    for (i in data.things) {
        x = data.things[i].thingName;

        var params3 = {
        topic: '$aws/things/'+x+'/shadow/update',
        payload: new Buffer('{"state": {"desired" : {"light" : "ON"}}}'),
        qos: 0
        };

       iotdata.publish(params3, function(err, data){
          ...
```

Fig. 7. Lambda function

5 Proposed Enhancements

In a typical ABAC model, attributes of the users (actors), who are requesting access, and attributes of the resources (target objects), on which accesses are performed, are utilized in the access control policies to determine allowed access on objects. The attributes in ABAC are name-value pairs and represent characteristics of entities, such as users and objects. Often environment or system attributes are also brought into consideration. In AWS IoT, things can have a set of attributes. The attributes are defined for virtual things in the cloud and are synchronized with their associated physical devices.

An example of thing attributes is shown in Fig. 8(a). Another way a thing can get attributes is through the certificate attachment or association as shown in Fig. 8(b). A number of attributes are set and defined while creating a X.509

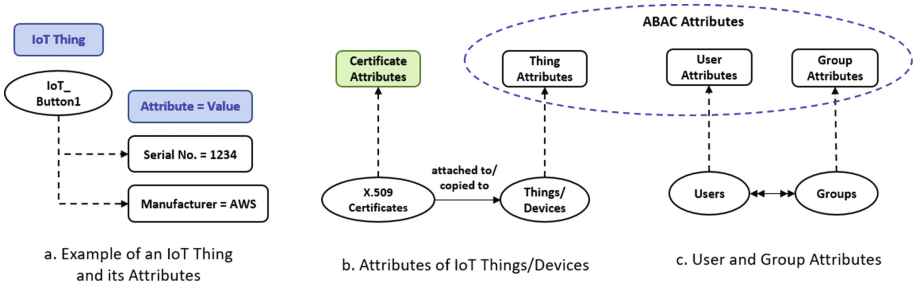


Fig. 8. Attributes in AWS IoT

certificate, and when a certificate is attached to a thing then the attributes of this certificate can be used in AWS IoT policies to assign permissions to the things. However, a certificate attribute does not reflect any direct properties of the thing it is attached to, and is thereby different than typical ABAC attributes.

Therefore, the access control model of AWS IoT (AWS-IoTAC) can be categorized as a restricted form of an ABAC model, mainly due to the following reasons.

- In AWS-IoTAC model, the attributes of only those IoT things/devices can be utilized which are requesting to perform actions on IoT resources (other IoT devices or applications) in the cloud.
- The thing attributes are applied in the policy only if the things/devices are using MQTT protocol to connect and communicate to the AWS IoT service.
- In AWS IoT, currently a thing can have only fifty attributes, of which only three are directly searchable.

Based on the above discussion and our exploration of the AWS IoT service, we propose some enhancements for the AWS-IoTAC model in order to incorporate a more complete form of ABAC in the model.

1. **ABAC Including Attributes of Target Resources**
   As discussed in our use case scenario 2, the AWS-IoTAC model should be able to incorporate attributes of things/devices performing IoT operations as well as attributes of things/devices on which the operations are being performed, independent of the connection and communication protocol being used. The target resource attributes are mainly useful in isolating the identity of specific IoT objects. For example, an IoT device needs to publish messages to other devices which have some specific attribute values. The publishing device need not be aware of the specific topics it need to publish to, and can publish to multiple topics meeting some specific criteria.
2. **ABAC Including User and Group Attributes**
   A more complete form of ABAC would require inclusion of attributes of users and groups of users, as shown in Fig. 8(c). In real-world IoT systems, there are multiple users using and controlling IoT devices. Therefore, including users

and devices relationships in access control decisions facilitates fine-grained authorization in cloud-enabled IoT platforms.

3. **Policy Management Utilizing the Policy Machine**
AWS offers a form of policy-based access control based on policy files attached to entities such as users, groups, "roles", and certificates. For all these entities, there are numerous policies defined. With billions of devices and their users, the policies for them will scale tremendously and soon become unmanageable. In near future, a possible problem that AWS might encounter is a policy-explosion problem. While setting up our use case as an administrator, we realized the need for a customer-based policy management tool. Policy Machine (PM) [13,14], an access control specification and enforcement tool developed by National Institute of Standards and Technology (NIST), could be utilized in this context. However, more detailed analysis of our model with respect to PM would be required to demonstrate its applicability.

6 Conclusion

We presented a formal access control model for AWS IoT. AWS is one of the largest cloud computing platforms that provides numerous services and products along with their extensive documentations. It was a challenge to incorporate all the aspects and capabilities of its IoT platform in our model. We mainly focused on access control and authorizations in a real cloud-enabled IoT platform. We believe our model would act as an initial blueprint for developing a generalized access control model for cloud-enabled IoT which can be incrementally enhanced to incorporate new IoT access control capabilities. We also proposed some enhancements to the AWS-IoTAC model based on our experience during use case setup and configuration. ABAC seems to be a promising access control model for the IoT services. For the future work, we will explore ways to incorporate the ABAC enhancements in our model, including both client (e.g., thing, user, application) attributes and target resource (e.g., things, applications) attributes. We also plan to investigate access control and authorizations in other real-world cloud-enabled IoT platforms.

Acknowledgments. This research is partially supported by NSF Grants CNS-1111925, CNS-1423481, CNS-1538418, and DoD ARL Grant W911NF-15-1-0518.

References

1. Amazon Web Services (AWS). https://aws.amazon.com/. Accessed 10 Dec 2016
2. AWS IoT Platform. http://docs.aws.amazon.com/iot/latest/developerguide/what-is-aws-iot.html. Accessed 8 Jan 2017
3. AWS SDK for JavaScript in Node.js. https://aws.amazon.com/sdk-for-node-js/. Accessed 10 Aug 2016
4. Azure IoT. https://docs.microsoft.com/en-us/azure/iot-hub/iot-hub-what-is-iot-hub. Accessed 10 Nov 2016

5. Build your blueprint for the internet of things, based on ve architecture styles. https://www.gartner.com/doc/2854218/build-blueprint-internet-things-based. Accessed 2 Jan 2017
6. Google Cloud Platform. https://cloud.google.com/. Accessed 10 Dec 2016
7. Microsoft Azure. https://azure.microsoft.com/en-us/. Accessed 28 Nov 2016
8. MQTT.fx - A JavaFX based MQTT Client. http://www.mqttfx.org/. Accessed 10 Sep 2016
9. Overview of Internet of Things. https://cloud.google.com/solutions/iot-overview/. Accessed 10 Dec 2016
10. X.509 Certificates. http://searchsecurity.techtarget.com/denition/X509-certificate. Accessed 10 Feb 2017
11. Al-Kahtani, M.A., Sandhu, R.: A model for attribute-based user-role assignment. In: 18th IEEE Annual Computer Security Applications Conference, pp. 353–362. IEEE (2002)
12. Alshehri, A., Sandhu, R.: Access control models for cloud-enabled internet of things: a proposed architecture and research agenda. In: 2nd IEEE International Conference on Collaboration and Internet Computing (CIC), pp. 530–538. IEEE (2016)
13. Ferraiolo, D., Atluri, V., Gavrila, S.: The policy machine: a novel architecture and framework for access control policy specification and enforcement. J. Syst. Archit. **57**(4), 412–424 (2011)
14. Ferraiolo, D., Gavrila, S., Jansen, W.: Policy Machine: features, architecture, and specification. NIST Internal Report 7987 (2014)
15. Ferraiolo, D.F., Sandhu, R., Gavrila, S., Kuhn, D.R., Chandramouli, R.: Proposed NIST standard for role-based access control. ACM Trans. Inf. Syst. Secur. (TISSEC) **4**(3), 224–274 (2001)
16. Gusmeroli, S., Piccione, S., Rotondi, D.: A capability-based security approach to manage access control in the Internet of Things. Math. Comput. Modell. **58**(5), 1189–1205 (2013)
17. Hernández-Ramos, J.L., Jara, A.J., Marin, L., Skarmeta, A.F.: Distributed capability-based access control for the Internet of Things. J. Internet Serv. Inf. Secur. (JISIS) **3**(3/4), 1–16 (2013)
18. Hu, V.C., Ferraiolo, D., Kuhn, R., Schnitzer, A., Sandlin, K., Miller, R., Scarfone, K.: Guide to attribute based access control (ABAC) definition and considerations. NIST Special Publication 800–162 (2014)
19. Jin, X., Krishnan, R., Sandhu, R.: A unified attribute-based access control model covering DAC, MAC and RBAC. In: Cuppens-Boulahia, N., Cuppens, F., Garcia-Alfaro, J. (eds.) DBSec 2012. LNCS, vol. 7371, pp. 41–55. Springer, Heidelberg (2012). doi:10.1007/978-3-642-31540-4_4
20. Kaiwen, S., Lihua, Y.: Attribute-role-based hybrid access control in the Internet of Things. In: Han, W., Huang, Z., Hu, C., Zhang, H., Guo, L. (eds.) APWeb 2014. LNCS, vol. 8710, pp. 333–343. Springer, Cham (2014). doi:10.1007/978-3-319-11119-3_31
21. Kuhn, D.R., Coyne, E.J., Weil, T.R.: Adding attributes to role-based access control. Computer **43**(6), 79–81 (2010)
22. Liu, J., Xiao, Y., Chen, C.P.: Authentication and access control in the Internet of Things. In: 32nd IEEE International Conference on Distributed Computing Systems Workshops (ICDCSW), pp. 588–592. IEEE (2012)

23. Mahalle, P.N., Anggorojati, B., Prasad, N.R., Prasad, R.: Identity establishment and capability based access control (IECAC) scheme for Internet of Things. In: 15th IEEE Symposium on Wireless Personal Multimedia Communications (WPMC), pp. 187–191. IEEE (2012)
24. Nitti, M., Pilloni, V., Colistra, G., Atzori, L.: The virtual object as a major element of the internet of things: a survey. IEEE Commun. Surv. Tutorials **18**(2), 1228–1240 (2016)
25. Ouaddah, A., Mousannif, H., Elkalam, A.A., Ouahman, A.A.: Access control in the Internet of Things: big challenges and new opportunities. Comput. Netw. **112**, 237–262 (2017)
26. Rajpoot, Q.M., Jensen, C.D., Krishnan, R.: Integrating attributes into role-based access control. In: Samarati, P. (ed.) DBSec 2015. LNCS, vol. 9149, pp. 242–249. Springer, Cham (2015). doi:10.1007/978-3-319-20810-7_17
27. Sandhu, R., Coyne, E.J., Feinstein, H., Youman, C.: Role-based access control models. Computer **29**(2), 38–47 (1996)
28. Zhang, G., Tian, J.: An extended role based access control model for the Internet of Things. In: IEEE International Conference on Information Networking and Automation (ICINA), vol. 1, pp. V1-319–V1-323. IEEE (2010)
29. Zhang, Y., Patwa, F., Sandhu, R.: Community-based secure information and resource sharing in AWS public cloud. In: 1st IEEE Conference on Collaboration and Internet Computing (CIC), pp. 46–53. IEEE (2015)