Context-aware and Attribute-based Access Control Applying Proactive Computing to IoT System

Noé Picard[1], Jean-Noël Colin[1] and Denis Zampunieris[2]
[1]*PReCISE Research Center, University of Namur, Belgium*
[2]*FSTC, University of Luxembourg, Luxembourg*

Keywords: Internet of Things, Access Control, ABAC, Event Analysis, Proactive Computing.

Abstract: ABAC allows for high flexibility in access control over a system through the definition of policies based on attribute values. In the context of an IoT-based system, these data can be supplied through its sensors connected to the real world, allowing for context-awareness. However, the ABAC model alone does not include proposals for implementing security policies based on verified and/or meaningful values rather than on raw data flowing from the sensors. Nor does it allow to implement immediate action on the system when some security flaw is detected, while this possibility technically exists if the system is equipped with actuators next to its sensors. We show how to circumvent these limitations by adding a proactive engine to the ABAC components, that runs rule-based scenarios devoted to sensor data pre-processing, to higher-level information storage in the PIP, and to real-time, automatic reaction on the system through its actuators when required.

1 INTRODUCTION

Access control is an important part of today systems. Whether it is about physical access control or software access control, it has become an essential and critical element for most businesses. Over the years, multiple models have emerged, like the relatively recent model called Attribute-based access control (ABAC). The concept is not especially innovative, but with the rise of Internet of Things (IoT) systems, the ABAC model could turn out to be very interesting. In fact, IoT sensors allow for monitoring and collecting data about the environment on which access control might apply. Moreover, IoT actuators provide a way for the system to interact with the physical world. With all the data that such a system can provide, the ABAC policies could only be enhanced (Rath and Colin, 2017).

However, some concerns arise when one tries to apply those two concepts together. To make the data coming from the sensors available to the ABAC model architecture, a gap between two components has to be filled, as we will demonstrate in Section 3, and no straightforward solution could be found. Ideally, the envisioned system would also interact with the surrounding environment. Still, how to make this interaction easier without implementing a completely new system? These are the issues that we want to address with the help of proactive computing. We introduce a way to use the proactive computing paradigm, through a proactive engine, to enhance the ABAC model with an IoT system.

The rest of this paper is structured as follows. In Section 2, the access control part is explained along the ABAC model. Section 3 describes the consequences of applying access control to an IoT system. Section 4 explains how the proactive computing makes this application easier. The resulting architecture, which derives from the ABAC model, is described in Section 5. To illustrate how the proactive engine would work in a such case, the Section 6 provides a direct use case example. Section 7 describes how it can be extended to real working systems. Finally, Section 8 completes this paper with our conclusions.

2 ATTRIBUTE-BASED ACCESS CONTROL

Access control is used to define what a subject can do or which resources it can access. Several models for access control have been proposed in the literature like, e.g., discretionary access control, mandatory access control or role-based access control. In this pa-

Picard, N., Colin, J-N. and Zampunieris, D.
Context-aware and Attribute-based Access Control Applying Proactive Computing to IoT System.
In *Proceedings of the 3rd International Conference on Internet of Things, Big Data and Security (IoTBDS 2018)*, pages 333-339
ISBN: 978-989-758-296-7

per, we are concerned with the Attribute-Based Access Control (ABAC) model. For an introduction to ABAC, see for example (Hu et al., 2014).

The main goal of ABAC is to provide high flexibility in access control. For instance, it considers that access control does not always only concern the identity and related roles of the subject trying to access a resource. Indeed, additional information could influence the access decision like the current state of the subject, its actions or the environment. These data are referenced as attributes in the ABAC terminology and attributes are the building blocks of access control policies. Within those policies, attributes can be combined in complex expressions.

2.1 ABAC Model Architecture

The underlying architecture to the ABAC model is generally composed of three main components. The Policy Enforcement Point (PEP) protects the services, data, etc. on which access control is required. When a subject wants to access a resource, the PEP intercepts this request and translates it in an authorization request understandable by the Policy Decision Point (PDP).

The PDP is at the core of the architecture as is the component that takes the final decision (Deny/Permit) regarding a request. This node often relies on two sub-components. One is the Policy Repository Point (PRP) which is responsible for making policies available to the PDP. The other one is the Policy Administration Point (PAP) that acts as the interface for system administrators that allows them to create, edit or delete policies.

Finally, the Policy Information Point (PIP) allows the PDP to retrieve the current attribute values that are needed for the computation of the policies. The rest of this paper is mainly concerned by discussions and proposals around the PIP.

2.2 XACML Standard

XACML (OASIS Standard, 2013) stands for eXtensible Access Control Markup Language. It has been created as a standard by defining a language and a protocol to convey information about access control. It is mainly based on ABAC, but it can also be specialized for other access control models like RBAC. The XACML language gives the possibility to define policies using XML notations. The standard proposes a computation model to evaluate policies against access requests which is based on the ABAC architecture described in Section 2.1.

3 APPLYING ACCESS CONTROL TO IoT

One of the strengths of ABAC is the possibility to use a set of multiple and diverse attributes in the policies. It is even more interesting if this set can be supplied with data from an IoT system, because it can provide lots of information through its sensors connected to the real world.

3.1 Main Benefits from this Combination

As a simple example, one could think of a room equipped with connected sensors that monitor temperature, humidity, presence, and so on in the room. If we store the data received by those devices in a database available through the PIP, then we could write XACML policies that allow or not the access to the room (supposing that there are also connected actuators that (un)lock the room doors, see Section 3.2 and 4) that rely on the current contextual information in the room.

A second additional benefit is the possibility to control how the database behind the PIP is supplied with flows of data. In XACML, there is no recommended way to implement this database, so it allows us to imagine a solution that processes the data coming from the sensors before to store information in the PIP. This way, by avoiding to use raw values from the sensors that may not be relevant and/or by aggregating several values into upper-level data, we could enhance the coherence and the pertinence of the attributes available to the set of policies.

3.2 Limits in Its Implementation

While ABAC offers great extensibility and flexibility compared to other access control models, developers will face some of its limits when it comes to implement it in a real-world system.

For instance, knowing how to fill the attribute values backing up the PIP might be as simple as inserting information in a database, but who is responsible to collect and store the data? It is not really a limit of the ABAC model that we underline here, as it takes into account the diversity of the attributes sources behind the PIP by using it as an interface between the sources and the PDP. But the concern is that if we heavily base the access policies on environmental attributes, it might be cumbersome to implement systems that fill the sources of the PIP.

Also, assuming one is using ABAC with an IoT system, it surely desires to interact with the environ-

ment through the sensors and the actuators. But there is no way to be able to interact synchronously with the environment just by using ABAC. One might say that there is the mechanism of obligations, defined in XACML as operations that should be performed by the PEP when they are returned alongside the authorization decision. But this only allows the PEP to have an effect on the environment when a subject tries to access a resource, not when something has changed within the environment.

4 PROACTIVE COMPUTING AT HELP

Before talking about how to address the limits laid down in Section 3.2, using proactive computing, let us first introduce it. In an article, Tennenhouse established theoretically the basics of proactive computing (Tennenhouse, 2000). He described it in regards with his vision of the future of computing, namely the transition from "human-centered to human-supervised (or even unsupervised) computing". In other words, human-centered computing can be referenced as interactive computing, where a system being used by a user is locked to wait for user actions. However, with proactive computing, the human is no longer placed in the loop, but above it.

4.1 Proactive Engine

At the University of Luxembourg, Zampunieris D. and his team had developed a proactive engine that follows the premises of proactive computing as described by Tennenhouse: "working on behalf of, or *pro*, the user, and acting on their own initiative". Without entering too much in the detail of the implementation, there are some key concepts to understand.

The engine is a *Rule-running system (RRS)*, it executes rules at a certain frequency. With the concepts of *Rule*, one can create a *Scenario* which is a dynamic set of rules obeying some path.

Furthermore, a *Rule* is the basic element in the engine and it can be subdivided in five stages. (1) The first one is called *Data acquisition*. It is during this stage that all the data, necessary to the proper execution of the following stages, is gathered. (2) The *activation guards* stage acts like a trigger for the third and fourth stages. It can be any type of boolean expression. (3) The purpose of the third stage, *Conditions*, is to offer the possibility to perform more in-depth tests on the context. (4) To effectively do something, there is the *Actions* phase, which provides a lot of freedom as well as power, given that the system is implemented in the Java programming language. (5) Finally, the *Rule Generation* step concludes the execution of a rule and allows the creation of other rules, or to clone itself to stay in the system.

To store the rules to be executed at each iteration (or at a further one if it is impossible to execute all of them in one iteration), the engine relies on a FIFO queue. For more information about the other aspects of the engine, see (Zampunieris, 2006).

4.2 Make the Combination Easier

In the Section 3.2, we laid down some limits of the ABAC model, especially when it is applied to an IoT system. These can be easily taken away using a proactive engine in an effective way. In fact, a proactive engine could fill the gap between an ABAC architecture and an IoT system.

On the one hand, there is the concern about the way to make attribute values, coming from sensors data, available to the PDP. Of course, it is the purpose of the PIP to make those available, but as stated before, it is just an interface between some data sources and the PDP. The idea is to use some rules (ideally multiple scenarios) to supply a particular source holding sensors data. But it would not just be raw data, as it is possible to implement scenarios that somehow (see Section 6 for an example) analyze, compute, combine, etc. this data. Therefore, we can assure a certain coherence and completeness in what we store as attributes.

Before going further, note that the scenarios must not be mistaken with Artificial Intelligence (AI), those are predefined, nevertheless they can be multiple and complex. Their sequencing is decided over a period of time based on the events that have occurred.

Then, on the other hand, proactive computing helps a lot when it is necessary to react on the actuators of the IoT system. Like before, specific scenarios could stay tuned for new events that require action(s) and react as quickly as possible.

5 ENHANCED ABAC MODEL ARCHITECTURE

To better understand how a proactive engine can handle his two main tasks (assigned in Section 4.2), this section explains the logical architecture that we came up with. In the Figure 1, there are 3 important parts: the access control part, the proactive engine and the IoT system.

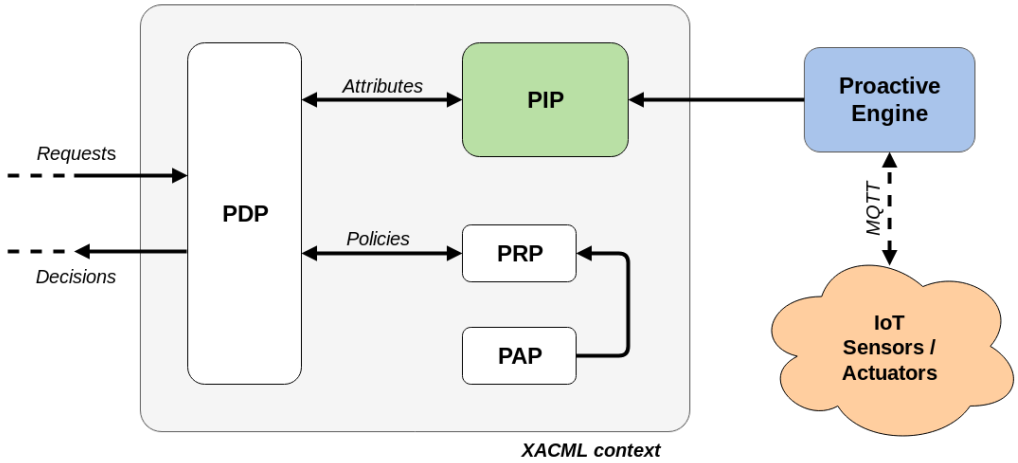

Figure 1: Enhanced ABAC model architecture.

5.1 Comply with the Standard

The access control part is delimited by the *XACML context* frame. It follows the architecture described in 2.1, the PDP receive the authorization requests and deliver the decisions. To do so, it retrieves the policies applicable to the current request and evaluates them. If the policies contains references to attributes not available in the direct context – which will be almost always the case because we want to rely exclusively on environmental attributes, the PDP asks the PIP for the corresponding values.

Note that the PEP component is intentionally not represented on the Figure 1, because usually there will be several of them and it can be in many forms and at different places across a system.

5.2 Enhanced Proactive Engine

The proactive engine described in Section 4.1 lacks of one important feature: the communication with the IoT system. Therefore, we chose to implement alongside the engine a MQTT broker. MQTT (Message Queuing Telemetry Transport) is a protocol based on the *Publish-subscribe* pattern, where senders do not know who they are sending messages to, but instead, publish messages on topics. Similarly, the receivers express their interest for a certain topic without knowing anything about the senders. For detailed information about the MQTT protocol, see (OASIS Standard Incorporating Approved Errata 01, 2015). In this way, a rule inside the engine can subscribe to a certain topic and thereby listen for new messages. This specific type of rule has a buffer where the messages can pile up to be processed afterwards.

Moreover, the proactive engine is connected to a database, implicitly represented by the PIP on the Fig. 1. This allows any rule to insert or retrieve data from it. The schema of the database is left to the developers, as it depends on the type of the information contained in the MQTT messages, and therefore the type of the sensors used.

5.3 Collect Data and Act on the Environment

The final important part is the IoT system (composed of sensors and actuators). As hinted in Section 5.2, the sensors can send their data (using specific topics) to the MQTT broker embedded in the proactive engine. The other way around, for actuators to receive commands, the rules can also send messages through the broker.

6 USE CASE EXAMPLE

To illustrate how our architecture works, this section includes a straightforward example. Consider a simple server room equipped with temperature sensors and a system to cool the place that we can utilize as an actuator. The scenario – from the proactive engine point of view – represented at the Figure 2 will monitor the temperature and take actions.

The sensors send their data to the topics `sensors/temperature/X`, where `X` is the identification number of the sensor. A rule in the proactive engine, named `SENSORS_TEMPERATURE_MONITORING` for example, could subscribe to the topic filter `sensors/temperature/+`, meaning that this rule

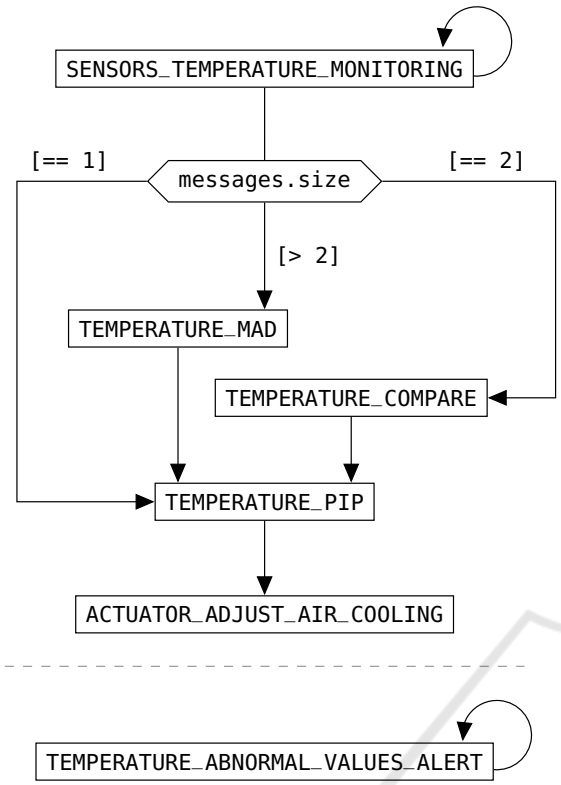

Figure 2: Use case example - Scenario structure.

will receive data from all temperature sensors. This rule will always analyse the latest twenty messages received, by clearing its messages buffer in the *actions* step if necessary. If the messages buffer is empty, the rule is not activated. If it has been, it creates other rules according to the number of messages. Note that this rule has a specific type or class inside the engine, we could call it a *MQTT rule*. This type of rule is automatically cloned in the queue in order to always listen for new MQTT messages. The following pseudo code describes its logic:

```
data acquisition: /

activation guards:
  messages.size() > 0

conditions: /

actions:
  if (messages.size() > N_TEMPERATURES_TAKEN
      )
    messages.subList(0, (messages.size() -
        1) - N_TEMPERATURES_TAKEN).clear()

  tempsToAnalyze = messages

rules generation:
  if (getActivated())
    size = tempsToAnalyze.size()
    if (size = 1)
      createRule(new TEMPERATURE_PIP(
          tempsToAnalyze.get(0)))
    else if (size = 2)
      createRule(new TEMPERATURE_COMPARE(
          tempsToAnalyze))
    else
      createRule(new TEMPERATURE_MAD(
          tempstoanalyze))
```

From the SENSORS_TEMPERATURE_MONITORING rule, there is one of the three following rules created. If only one message was received, the created one is the TEMPERATURE_PIP rule. Its purpose is to store in the PIP the temperature value passed as a parameter. To do so, it verifies if it is necessary to store the value, i.e. if the difference between the current stored value in the PIP and the value to store is greater than a particular constant (MINIMUM_DELTA). If not, the rule is not activated and nothing is stored. Moreover, if the difference is greater than another constant (MAXIMUM_DELTA) and that the last temperature update in the PIP was close enough, the value is considered as abnormal and a counter is incremented. This rule is an example of how the proactive engine can solve the first limit presented in Section 3.2. In the final step, the rule ACTUATOR_ADJUST_AIR_COOLING is created to take actions on the environment through the cooling system by triggering some actuators.

```
data acquisition:
  previousTemp = getPip().getTemperature()
  lastTempUpdate = getPip().
      getLastTemperatureUpdate()

activation guards: /

conditions:
  Math.abs(currentTemp - previousTemp) >
      MINIMUM_DELTA

actions:
  if ((lastTempUpdate + INTERVAL) >
      currentTime() && Math.abs(currentTemp
       - previousTemp) > MAXIMUM_DELTA)
    getEngineDB().
        incrementAbnormalTemperature
        ValuesCounter()

  getPip().updateTemperature(temperature)

rules generation:
  createRule(new ACTUATOR_ADJUST_AIR_COOLING
      (temperature))
```

The rule ACTUATOR_ADJUST_AIR_COOLING adjusts the air-cooling system according to the temperature value passed as parameter. Here, one can observe that this rule solves the second problem

developed in Section 3.2 by acting on the environment.

```
data acquisition: /

activation guards: /

conditions: /

actions:
  if (temperature < LIMIT_MIN) {
    setAirCoolingLevel(AirCoolingLevel.LOW)
  } else if (temperature > LIMIT_MAX) {
    setAirCoolingLevel(AirCoolingLevel.HIGH)
  } else {
    if ((temperature - LIMIT_MIN) <= 2) {
      setAirCoolingLevel(AirCoolingLevel.LOW
          )
    } else if (Math.abs(temperature -
        LIMIT_MAX) <= 2) {
      setAirCoolingLevel(AirCoolingLevel.
          HIGH)
    } else {
      // Between LIMIT_MIN and LIMIT_MAX
      setAirCoolingLevel(AirCoolingLevel.
          MODERATE)
    }
  }

rules generation: /
```

The rule TEMPERATURE_COMPARE is created by the rule SENSORS_TEMPERATURE_MONITORING when the messages buffer has a size of 2. It compares two temperatures values passed as arguments. If the difference between the two is not too high, the mean is store. But it can also increment the abnormal value counter if this difference is in fact too high.

When there are strictly more than 2 messages in the buffer, it is the rule TEMPERATURE_MAD that is created. This one uses a statistics measure called Median Absolute Deviation (MAD) to detect any abnormal value. If one is actually detected, as before, a counter is incremented.

These two last rules create the TEMPERATURE_PIP when it is necessary to store a temperature value in the PIP. Finally, the rule TEMPERATURE_ABNORMAL_VALUES_ALERT is always present in the engine queue, because it watches over the abnormal value counter and generates an alert if this counter exceeds some threshold.

7 INNOVATIVE, PROACTIVE, ABAC-BASED SECURITY

The example in Section 6 might seem simplistic, but it was essential to introduce and illustrate the main idea of this paper. However, it does not represent all the extent of the model that we developed. In fact, extending it to a real working system is achievable and has been already performed in a controlled testbed.

At the University of Luxembourg, a simulator representing the IoT system of a “modular data center” had been conceived to test a proactive engine with multiple scenarios. It was mainly simulating the sensors and the actuators of the system, to provide a kind of sandbox in order to experiment with the proactive engine. The results were conclusive, and prove that the engine could work on larger system, due to its ability to run several scenarios in parallel, in order to handle multiple sensors and actuators.

More globally, the advantages of our enhanced ABAC-based security model become clearer: First, as seen in Section 6, the proactive engine can take the role of filling automatically the PIP with sensors data. The access policies can be more abstract, that is to say the data stored as an attribute could have been pre-processed, thus it is not necessary to be burdened with low-level data in those policies. Secondly, the actuators can be triggered spontaneously by the scenarios, all it requires is implementing rules that interact with any of the actuators. Moreover, the coherence of attributes can also be insured if the scenarios are well-defined. Finally, another key aspect is the separation of concerns that allows the division in scenarios. With this concept it is possible to focus on one type of data or actuators at a time, which also allows a better maintainability and extendability.

8 CONCLUSIONS

Attribute-based access control applied to IoT system can theoretically make a powerful combination. To make it concrete, two limits were encountered: how to effectively link the sensors data with attribute values in the PIP and how to automatically and immediately act on the environment when it is needed. In order to overcome these, the proactive computing paradigm reflected in a proactive engine turned out to be an ideal choice.

With the concept of proactive scenario, we showed how to provide data coming from the sensors as a source of attributes for the ABAC model. Moreover, in these scenarios, the data can be pre-processed and not just be raw values that a sensor provides, hence allowing higher level data as attributes. Equally important is the ability to take actions on the environment by sending commands from the proactive engine to the IoT actuators.

This way of doing can be applicable to large sys-

tems with numerous sensors and actuators, as the engine can run multiple and complex scenarios in parallel, without performance issues. We believe that this new application of proactive computing opens some promising road towards taking full advantage of the ABAC model and the Internet of Things at the same time.

REFERENCES

Hu, C. T., Ferraiolo, D. F., Kuhn, D. R., Schnitzer, A., Sandlin, K., Miller, R., and Scarfone, K. (2014). Guide to Attribute Based Access Control (ABAC) Definition and Considerations. *Special Publication (NIST SP) - 800-162*.

OASIS Standard (2013). eXtensible Access Control Markup Language (XACML) Version 3.0. `http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-os-en.html`.

OASIS Standard Incorporating Approved Errata 01 (2015). MQTT Version 3.1.1 Plus Errata 01. `http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/mqtt-v3.1.1.html`.

Rath, T. M. A. and Colin, J. N. (2017). Adaptive Risk-aware Access Control Model for Internet of Things. In *Proceedings of the International Workshop on Secure Internet of Things, in conjunction with Esorics2017*, Oslo.

Tennenhouse, D. (2000). Proactive Computing. *Communications of the ACM*, 43(5):43–50.

Zampunieris, D. (2006). Implementation of a Proactive Learning Management System. In *Proceedings of "E-Learn - World Conference on E-Learning in Corporate, Government, Healthcare & Higher Education"*, pages 3145–3151.