HAL
open science

dynSMAUG: A Dynamic Security Management Framework Driven by Situations

Romain Laborde, Arnaud Oglaza, François Barrère, Abdelmalek Benzekri

To cite this version:

Romain Laborde, Arnaud Oglaza, François Barrère, Abdelmalek Benzekri. dynSMAUG: A Dynamic Security Management Framework Driven by Situations. 1st Cyber Security in Networking Conference (CSNet 2017), Oct 2017, Rio de Janeiro, Brazil. pp.1-8. hal-03656827

HAL Id: hal-03656827

https://hal.science/hal-03656827v1

Submitted on 2 May 2022

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Open Archive Toulouse Archive Ouverte

OATAO is an open access repository that collects the work of Toulouse researchers and makes it freely available over the web where possible

This is an author's version published in:
http://oatao.univ-toulouse.fr/22014

Official URL
https://doi.org/10.1109/CSNET.2017.8241987

To cite this version: Laborde, Romain and Oglaza, Arnaud and Barrère, François and Benzekri, Abdelmalek *dynSMAUG: A Dynamic Security Management Framework Driven by Situations.* (2017) In: 1st Cyber Security in Networking Conference (CSNet 2017), 18 October 2017 - 20 October 2017 (Rio de Janeiro, Brazil).

Any correspondence concerning this service should be sent to the repository administrator: tech-oatao@listes-diff.inp-toulouse.fr

dynSMAUG: A dynamic security management framework driven by situations

Romain Laborde, Arnaud Oglaza, François Barrère and Abdelmalek Benzekri
University Paul Sabatier
118 Route de Narbonne, F-31062 TOULOUSE CEDEX 9, France
Email: {Romain.Laborde, Arnaud.Oglaza, Francois.Barrere, Abdelmalek.Benzekri}@irit.fr

***Abstract*—We present a dynamic security management framework where security policies are specified according to situations. A situation allows to logically group dynamic constraints and make policies closer to business. Situations are specified and calculated by using complex events processing techniques and security policies are written in XACMLv3. Finally, the framework is supported by a modular event based deployment infrastructure. The whole framework has been implemented and its performance is evaluated.**

I. Introduction

The mission of IT security teams in organizations that consisted in *Protect to minimize risk* has shift to *Protect to enable* [1]. Security professionals should stop focusing on locking down assets advocating that priority is security. This approach, which constrains employees, is counterproductive to organizations. Security has to adapt itself to propose solutions that enable new business activities and new ways of working. In other words, the ideal security must allow organizations to grow in a secure way.

New technologies might enhance the employees' productivity, accelerate business processes and represent uncharted business opportunities. However, new technologies can also bring new threats and risks requiring additional security measures. In parallel, new threats and new vulnerabilities are reported continuously on the media. Consequently, security has to evolve on an ongoing basis. Thus, the security enforcement process should be flexible to quickly adapt the organization security to new usages/technologies and new threats.

To cope with new usages/technologies requirements, security management systems have to automate the enforcement of security measures. Mobility of employees or short term projects enhance the business agility. However, allowing only the right people to access the right assets under the right conditions (temporal, geographical, circumstances, etc) entails considering dynamic constraints that leads to dynamic security permissions.

In this article, we present *dynSMAUG* , our dynamic security management framework. The specificity of our approach consists in considering the concept of *situation* as the cornerstone of security management and we write policies herewith "when *situation* and *conditions* then *security actions*". On the one hand, situations allow to capture the dynamic constraints (time, location, etc.) and organize them into a stable and logical concept. Situation oriented security policies are simpler and more readable. Also, managing high level policies, close to business, reduces the gap between security requirements and the effective security policy executed by security devices, and then limits the security policy translation errors. On the other hand, making security policy more independent from technical constraints minimizes the impact of changing security mechanisms and simplifies the policy life cycle management. Situations are calculated by applying Complex Event Processing (CEP) techniques. CEP provides rich operators for describing complex situations that combines data from multiple sources. Policies are specified in XACMLv3. This standard implements the Attribute Based Access Control paradigm which makes the language highly flexible. In addition, this language supports both authorization and obligations. Finally, dynSMAUG includes a deployment infrastructure that dynamically enforces security policies. Its modular architecture facilitates the integration of new security mechanisms.

The rest of the article is structured as follows. Section II describes the problem highlighting the limits of security policy languages and architectures. In Section III, we present our proposition. We propose our definition of the word *situation* and we detail the specification of situations and situation based security policies. Section IV shows the deployment architecture and the performance evaluation of its implementation. Section V summarizes the related works. Finally, Section VI concludes and provides some perspectives for the future.

II. Description of the problem

In order to clearly state the problem, we first summarize the different existing approaches for managing security. Then, we present a scenario showing the limitations of these approaches.

A. Security management approaches

Policy based management (PBM) is a recognized and well established approach for managing complex systems now. One of the key motivations of this approach is flexibility and adaptability to existing infrastructure and change management. Policies represent an externalized logic that can determine the behavior of managed systems [2]. The architecture supporting this approach consists of a policy decision point (PDP) that interprets the policy and takes decision based on it, and a policy enforcement point (PEP) that compels the managed system to execute the decisions of the PDP.

If these generic concepts are common to all PBM solutions, two modes of deployments exist [3] which differ in how policies are specified and how the PEP and the PDP interact (See figure 1). The *outsourcing mode* is mainly implemented by authorization management systems. When a protected resource is being accessed, the PEP catches this access attempt and sends a request to the PDP. Based on its policies, the PDP takes its decision and transmits it to the PEP. The later applies the decision by granting this requested access or blocking it. In this deployment mode, interactions between the PEP and the PDP are synchronous. The PDP always sends a decision to a PEP as a transaction reply to a specific request. The policy languages of outsourcing mode systems are of the form '*condition* then *permit/deny*'. Examples of such systems are PERMIS [4], XACMLv3 [5].

On the other hand, the *provisioning mode* is more suitable for deploying security configurations. In this mode, the PDP receives an event from any sensor that triggers the decision-making process. The PDP can then transmit its decision to any PEP. Here, the interaction is asynchronous since the PEP can receive a decision from the PDP without having previously requested it. The policy languages of provisioning mode systems are of the form 'On *event* if *condition* then *action*'. PONDER [6] is an example of such systems.

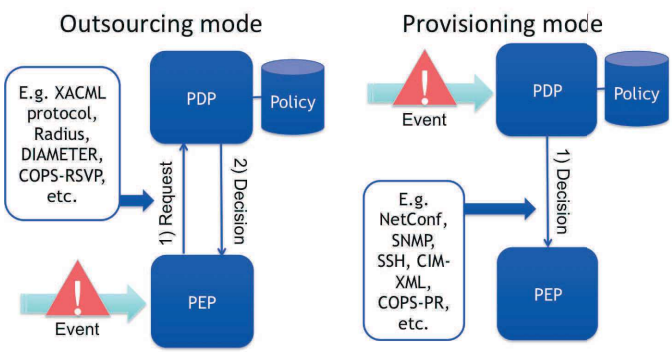

Fig. 1. Policy-Based management: Outsourcing and Provisionning modes

B. Scenario

We present in this section the scenario used in the rest of the article. It consists of a company working with sensible data that wants to prevent them from being compromised. The Information System Security Policy (ISSP) is the following : *Only owners can use their assigned computer. Whenever the owner of a device is not behind his/her computer, then the session shall be locked. The owner of a device is the employee who has been assigned to this device*. Since employees sometimes forget to lock their session and put sensible data at risk of being disclosed, the company decided to automate this task.

The enterprise proposes a solution to improve the policy by locating both users and computers. Consequently, when the owner is moving far from his associated device (e.g., the user exits the room), the session must be automatically locked. In the next sections, we call this rule *REACTIVE-RULE*. In addition, if someone tries to connect to the computer when the owner is far from his device, the connection must be refused and a notification message (SMS, email, etc.) must be sent to the owner. In the rest of the article, this rule is called *DENY-AUTHZ-RULE*. Finally, connection is allowed if the login/password is valid and the owner is in front of his device (rule *PERMIT-AUTHZ-RULE*).

In a first implementation, the company decided to locate every computer based on its GPS chip that regularly sends its coordinates. After agreeing with the employees upon privacy issues, the company also chooses to use the GPS system embedded in their smartphones. Using this tracking solution, *REACTIVE-RULE*, *DENY-AUTHZ-RULE* and *PERMIT-AUTHZ-RULE* can be rephrased as following:

REACTIVE-RULEv1
: *if* the distance between the positions of the owner and his computer becomes greater than X meters *then* lock the session

DENY-AUTHZ-RULEv1
: *if* (the connection to a computer is authenticated) *and* (the distance between the positions of the computer and its owner is greater than X meters) *then* refuse the connection *and* send an alert to the owner

PERMIT-AUTHZ-RULEv1
: *if* (the connection to a computer is authenticated) *and* (the distance between the positions of the computer and its owner is less than X meters) *then* permit the connection

After a testing period, the company realized that the GPS signal is not correctly received in all its premises. In addition, the computer can be within X meters of its owner and not within sight (e.g., there is a wall between the computer and the user, or they are on different floors). Thus, the first mechanism poorly implements the ISSP. To improve the security solution, the company decided to use an indoor positioning system that combines GPS, Wi-Fi, Bluetooth Low Energy, and motion sensors. This system is able to determine in which room computers and users are. Being more precise than the basic GPS positioning, the indoor positioning system provides the required accuracy to achieve the ISSP. However, introducing this new technology impacts the expression of *REACTIVE-RULE*, *DENY-AUTHZ-RULE* and *DENY-AUTHZ-RULE* that become:

REACTIVE-RULEv2
: *if* (the owner and his computer are in the same room) *and* (the owner or the computer is leaving the room) *then* lock the session

DENY-AUTHZ-RULEv2
: *if* (the connection to a computer is authenticated) *and* (the owner is not in the same room as that computer) *then* refuse the connection *and* send an alert to the owner

PERMIT-AUTHZ-RULEv2
: *if* (the connection to a computer is authenticated) *and* (the owner is in the same room as that computer) *then* permit the connection

This example highlights the two following issues:

Issue 1 : We refined each ISSP rule into two different versions, one version for each position tracking technology. In other words, the final security rules depend on the technology. New security rules will be added each time a new security related-technology is introduced. This will result in an unmanageable security policy. The security policy shall not depend on technology.

Issue 2 : Enforcing this ISSP requires a security management architecture that supports both outsourcing and provisioning modes. On the one hand, *REACTIVE-RULE* requires the management system to dynamically deploy a new security measure on the computers (*lock the session*) when an external event is received (*the owner is moving far away*). On the other hand, the outsouring mode fits with *DENY-AUTHZ-RULE* and *PERMIT-AUTHZ-RULE*.

In the next section, we propose our solution for specifying in a unified way technology independent security policies.

III. Expression of security policies driven by situations

We propose to make security policies independent from security mechanisms by specifying them according to situations. Our general idea consists in describing how situations arise on one side and specifying security policies oriented by situations on the other side. Both tasks can be performed in parallel.

A. Definition of situation

Although the word *situation* is commonly employed in ordinary, legal or even technical domains, clearly defining it is difficult and many definitions have been published [7]. Especially, *situation* is often interchanged with the word *context*. We present in this section our interpretation of the concept of situation and its differences with the concept of context.

Words *situation* and *context* are frequently employed in the domain of pervasive computing when dealing with context-awareness. Dey [8] has proposed one of the most popular definitions of context in this community. According to him, "*context is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves.*" This definition highlights some characteristics of the relation between *context* and *situation*: *context* is used to determine *situation*. Hence, *situation* is something more abstract than *context*. Dey has also introduced one definition of *situation* as "*The situation abstraction is [...] a description of the states of relevant entities.*".

Situational awareness is another domain that studied the concept of situation. Endsley [9] defines situational awareness as *the perception of the elements in the environment within a volume of time and space, the comprehension of their meaning, and the projection of their status in the near future.* This definition stresses two other points: situation is related to multiple entities and understanding situation means being able to project in the future.

Finally, Barwise and Perry [10] wrote on situation: *The world consists not just of objects, or of objects, properties and relations, but of objects having properties and standing in relations to one another. And there are parts of the world, clearly recognized [...]. These parts of the world are called situations.*

As a consequence, from a security management point of view, *context* is any instantaneous, detectable, and relevant information of the managed environment. The managed system is the world as defined in [10] and the context is all perceived information representing this world. More concretely, context is any data collected by sensors such as monitoring systems, intrusion detection systems, configuration management databases, etc. *Situation* is the result of computing relationships between parts of the managed entities included in the known context. The created relationships focus on one or all of the classic questions: who, when, where, why, what and how. We call the managed entities of interest as the *target of the situation*. In addition, situations convey meaning and provide abstraction of the context. They allow decision-making entities to understand what is happening and what could happen in the future. Thus, situations facilitates the decision making.

Our scenario in section II-B incorporates two situations: the user is close to his/her computer and (s)he is far from it. These two situations characterize a geographical relation between a user and his computer (Figure 2). In addition, these situations are closely related to each other forming a directed graph. We say both situations are in the same *situation class*. We propose the concept situation class to group situations logically. All situations of the same class must have the same situation target that constitutes the focus of the situation class. The other entities, which don't appear in all situations, are part of the context of the situation target and may change between situations of the same class.

In addition, describing the dynamics of the context as graph of situations facilitates the specification of security policies. For instance, the scenario ISSP is directly translated to:

- When the situation is *close*, the user can connect to his computer (PERMIT-AUTHZ-RULE);
- When the situation is *far*, nobody can connect to the computer and if someone tries to log in, the owner should be notified (DENY-AUTHZ-RULE);
- When the situation is moving from *close* to *far*, the session of the computer should be locked automatically (REACTIVE-RULE).

How situations *close* and *far* are calculated depends on the sensors and data available for computing the context.

B. Specification of situations

A situation is a particular time frame of interest that has a beginning, a life span and an end [11]. The beginning and the end of a situation are determined by combining multiple events coming from multiple sensors and occurring at different moments. Indeed, a situation involves multiple entities and

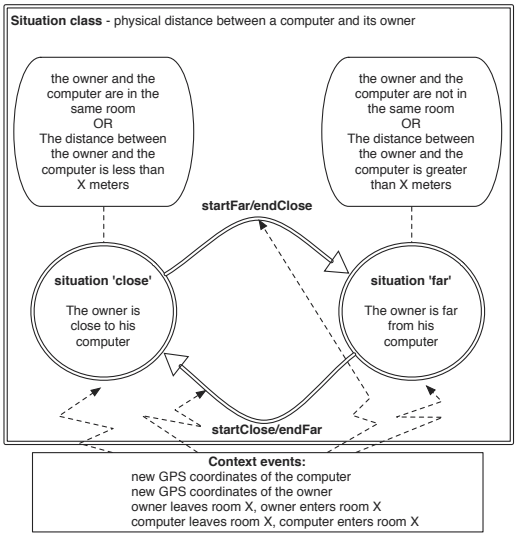

Fig. 2. Situations in our scenario

multiple conditions. The beginning and the end of a situation cannot be simple events captured by a unique sensor. In addition, events being instantaneous, combining multiple events requires complex temporal operators (event ordering, event existence/absence, time windows, etc.) to specify the beginning and end of situations.

In our scenario, the beginning and the end of situations *close* (referring to 'the user is close to his/her computer') and *far* (referring to 'the user is far from his/her computer') should be computed based on events related to both the user and the computer (see Figure 2). In addition, different combinations of events can determine the beginning and the end of the same situation. For instance, situation *far* starts when two events *GPS position of the user* and *GPS position of the computer* indicate that the distance between both protagonists is greater than X meters. Situation *far* also starts when applying a function on the events signalling that entities are entering or leaving rooms results in determining that the computer and its owner are located in two different rooms.

We follow the Complex Event Processing (CEP) approach for computing the beginning and the end of situations. CEP is "*a defined set of tools and techniques for analyzing and controlling the complex series of interrelated events that drive modern distributed information systems*" [12]. CEP solutions allow to specify complex events through complex event patterns that match incoming event notifications on the basis of their content as well as some ordering relationships on them. Cugola and Margara [13] have published an extended survey of CEP solutions.

We choose the open source event processing implementation called Esper, which is maintained by Espertech[1]. For specifying complex event patterns, Esper offers a stream-oriented language called Event Processing Language (EPL) that is an extension of SQL for processing events. EPL includes all the classical operators of SQL, as well as new features for windows definition and interaction, for timed-data arithmetic definition, and for complex event output generation.

In our scenario, situation *close* begins when situation *far* ends (Figure 2). Since complex events *startFar* and *startClose* are respectively similar to *endClose* and *endFar*, identifying situations *close* and *far* consists in expressing in EPL the two complex events *startFar/endClose* and *startClose/endFar*.

```
1   startFar = select * from
2   Phone-GPS_Event.win:time(6 sec)
3       as phone,
4   Laptop_GPS_Event.win:time(6 sec)
5           as laptop,
6   Active_Situation.std:lastevent()
7           as current-situation
8   where
9   phone.owner = laptop.owner and
10  gpsDistance(
11      phone.long, phone.lat,
12      laptop.long, laptop.lat) > 10  and
13  current-situation.situation-class =
14        "distance-between-laptop-and-owner" and
15  current-situation.situation-target =
16          laptop.owner and
17  not current-situation.value = "far"
```

Fig. 3. Situation "far" using GPS

The specification of the beginning and the end of situations depends on the input events and the characteristics of the sensors. Figure 3 is the EPL description of *startFar* of the first implementation of the ISSP, i.e. when the location of users and computers is determined by their GPS positions only. In our example, GPS sensors send the GPS position every 10 seconds. This EPL rule states that *1) in the last time window of 6 seconds, there is an event representing the GPS position of a smartphone (lines 2-3) and another for the position of a computer (lines 4-5) such that they belong to the same owner (line 9) and the distance is greater than 10 meters (lines 10-12) while the last current situation is not far (lines 13-17)*. Since GPS sends coordinates every 10 seconds, we are sure to receive the coordinates of both the user and the computer within 6 seconds and then we can limit the time window for comparing GPS coordinates to this period.

In the second implementation, which consists in the indoor positioning system, indoor position sensors trigger events when smartphones or computers are entering in a new room. This changes the way that complex event *startFar* is calculated to either the user or the computer has moved to another room while both were initially in the same room. The indoor positioning system provided by PoleStar used in project Box@PME takes between 0 to 10 seconds to calculate the position of a device[2]. Therefore, there may be a delay of 10 seconds between the time a device actually moves to another room (the world has changed) and the time the event reporting this change is actually received (the context is actually updated). Thus, situation *far* starts when there is

[1] http://www.espertech.com/esper/

[2] this constraint of 10 seconds is specific to our lab experimentation

event *computer enters room X* (or *smartphone enters room X*) while the current situation is *close*, and no event *smartphone enters room X* (or *computer enters room X*) is triggered within 10 seconds. To translate this statement in EPL, we specify event *startFar* in two steps (see Figure 4). First, we compute two complex events: *i*) *MayBeFarEvent* states either the smartphone (lines 4-5) or the computer (lines 6-7) of the same owner (line 11) moves to another room (line 12) while the current situation is *close* (lines 13-17), and *ii*) *MayBeCloseEvent* when the smartphone (lines 22-23) or the computer (lines 24-25) moves to a room such that both devices are then in the same room (line 30) and while the current situation is *far* (lines 31-35). These complex events are not the beginning or end of a situation. They will be re-injected in the CEP system. The beginning of situation *far* can be computed when an event *MayBeFarEvent* has been triggered (line 38) and after waiting 10 seconds (line 39) no event *MayBeCloseEvent* appears for the same owner (line 40).

```
1   maybeFarEvent = insert into MayBeFarEvent
2   select phone.owner as owner, phone, laptop
3   from
4   Phone_Geofencing_Event.std:unique(owner)
5           as phone,
6   Laptop_Geofencing_Event.std:unique(owner)
7           as laptop,
8   Active_Situation.std:lastevent()
9           as current-situation
10  where
11  laptop.owner = phone.owner and
12  not phone.location = laptop.location and
13  current-situation.situation-class =
14        "distance-between-laptop-and-owner" and
15  current-situation.situation-target =
16          laptop.owner and
17  current-situation.value = "close"
18
19  maybeCloseEvent = insert into MayBeCloseEvent
20  select phone.owner as owner, phone, laptop
21  from
22  Phone_Geofencing_Event.std:unique(owner)
23          as phone,
24  Laptop_Geofencing_Event.std:unique(owner)
25          as laptop,
26  Active_Situation.std:lastevent()
27          as current-situation
28  where
29  laptop.owner = phone.owner and
30  phone.location = laptop.location and
31  current-situation.situation-class =
32          "distance-between-laptop-and-owner" and
33  current-situation.situation-target =
34          laptop.owner and
35  current-situation.value = "far"
36
37  startFar = select * from pattern [
38  every(mayBeFarEvent=MayBeFarEvent ->
39  timer:interval(10sec) and
40  not MayBeCloseEvent(owner=mayBeFarEvent.owner))]
```

Fig. 4. Situation "far" using indoor positioning

C. Specification of situation-Based security policies

In our approach, situations are specified outside the security policy. Therefore, the security policy can refer to them only without requiring to describe them. As a result, we represent security policies in a generic way as : when *situation* and *some condition* then *authorization decision and/or obligation(s)*. As highlighted in section II-B, the security policy language shall allow the security administrator to specify both reactive and authorization rules. These two kinds of rules can be easily written following these patterns:

reactive rules : when *situation* and *situation begins* [and *some condition*] then *obligation(s)*

authorization rules : when *situation* and *some condition about the requested access* [and *some other condition*] then *authorization decision and/or obligation(s)*

We propose to express our security policies in XACMLv3 [5], which is standardized by the OASIS. First, it follows the Attribute Based Access Control (ABAC) approach [14] where policies describe general access control requirements in term of constraints on security attributes; attributes being any characteristics of entities. Hence, "the rules or policies that can be implemented in an ABAC model are limited only to the degree imposed by the computational language" [14]. Initially, XACML policy only considered attributes of the subject, the resource, the action and the environment. However, OASIS resolved this issue since version 3 and it is now possible to consider any managed entities thanks to the concept of categories. In addition, the XACMLv3 policy language includes *obligations*. Thus, XACMLv3 is not limited to PERMIT/DENY decisions only and can also describe complex decisions involving the modification of managed entities. Finally, the XACMLv3 language is *extensible* [15], [16].

However, XACML is a verbose XML language which makes difficult to write or read security policies [17]. Abbreviated Language for Authorization (ALFA) [18] has overcome this issue. ALFA provides the means to present XACMLv3 policies in compact forms. It does not increase nor reduce the semantics of XACMLv3 but provides human-readable policies. The translation between ALFA and XACMLv3 must be explicitly defined in mapping files.

Figure 5 shows the three security rules of our example in ALFA. It is worth to notice that this policy in ALFA is close to the original ISSP stated in section II-B, which limits the errors when translating a high level security policy into a target security policy language. The rule called *reactive_rule* means: when *the value of situation class Situation_laptop_owner is far* (line 4) and *the situation begins* (line 5) then *obligation: lock the session of the related user* (lines 7-10). Attribute *recipient* in the obligation (line 8) indicates where to enforce the obligation. If attribute *recipient* is not filled, the obligation is returned to the management entity that requested a decision. The second rule *deny_authz_rule* involves three entities: a situation, a resource and an action. It states: when *the value of situation class Situation_laptop_owner is far* (line 17) and *an authenticated connection on the resource has been performed* (line 18) then *the authorization decision is deny* (line 15) and *obligation: send a notification on the smartphone of the owner* (line 19-23). Finally, the last rule allows access (line 28) when *the value of situation class Situation_laptop_owner is close*

(line 30) and *the connection is authenticated* (line 31).

```
1  rule reactive_rule {
2      permit
3    target clause
4      Situation_laptop_owner.value == "far"
5      and Situation_laptop_owner.state=="begin"
6    on permit{
7      obligation Obligations.lockSession {
8        Attribute.recipient=
9           Situation_laptop_owner.situation-target
10       Attribute.device="laptop"}
11   }
12 }
13
14 rule deny_authz_rule {
15   deny
16   target clause
17     Situation_laptop_owner.value == "far"
18     and Action.name=="authenticated-connection"
19   on deny{
20     obligation Obligations.notifyUser {
21       Attribute.recipient=
22          Situation_laptop_owner.situation-target
23       Attribute.device="smartphone" }
24   }
25 }
26
27 rule permit_authz_rule {
28   permit
29   target clause
30     Situation_laptop_owner.value == "close"
31     and Action.name == "authenticated-connection"
32 }
```

Fig. 5. Our example ISSP Policy in ALFA

IV. Enforcement of Security Policies Driven by Situations

Now, we present the deployment architecture of dynSMAUG and its performance evaluation.

A. The deployment infrastructure

The architecture of dynSMAUG aims at allowing the deployment of security policies in both outsourcing and provisioning modes (Figure 6). In that way, we first split the PEP entity into its two basic functionalities: the sensor role that captures events in the managed system and the actuator role that enforces policy decisions. The PEP in the outsourcing mode (Figure 1) plays both roles: it is a sensor when it sends a decision request to the PDP and an actuator when it enforces the returned decision. In the provisioning mode, it only plays the role of actuator while the management entity that detected the event played the role sensor. Secondly, we consider every messages as events to unify the interactions between the sensors/actuators and the PDP. Hence, the protocol in the outsourcing mode consists in two events (one for the request and the other for the decision) while the protocol in the provisioning mode is a unique event (i.e., the decision).

The actors of our deployment architecture are shown in Figure 6:

- The *broker* is the distribution middleware that transmits all the events between the actors following the publish-subscribe pattern. The broker divides events into three topics: the context events, the situation events and the decision events.
- The *sensors* produce data events of the context (noted c in Figure 6). A context event can be the GPS coordinates, the user exiting a room, or an attempt to access a protected resource. An event is defined in XACMLv3 by a set of attributes of the form <*identifier, type, value*>. This solution has an advantage: it is possible to develop sensors in any programming language.
- The *situation manager* contains a complex event processing engine that calculates situations according to a situations specification as explained in section III-B. It consumes context events and produces situation events (noted *s* in Figure 6). Situation events have the same format as context events and are also carried in XACMLv3 requests. Each time a new situation is calculated, the situation manager creates two situations events: the first event informs the beginning of the new situation and the second one notifies the end of the last active situation.
- The *command center* is the brain of our security management framework. It consumes both context and situation events and produces decision events (noted *d* in Figure 6). As explained in section III-C, we specify security policies in XACMLv3. However, the XACML PDP only implements the outsourcing mode. Therefore, the command center includes an XACMLv3 PDP and a decision proxy. The main objective of the decision proxy is to allow the command center to operate both outsourcing and provisioning modes. The context and situation events are received by the decision proxy. Then, the decision proxy transmits the events to the PDP and wait for the decision.

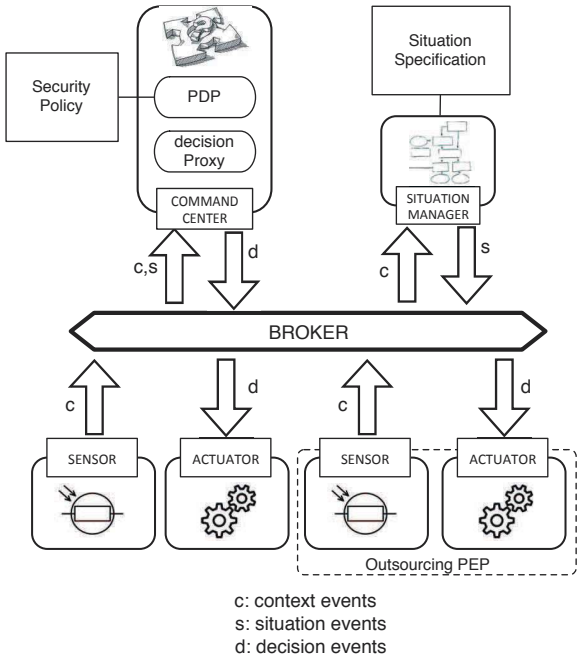

Fig. 6. Architecture of dynSMAUG

The PDP acts in compliance with XACMLv3. When the decision proxy receives the decision from the PDP, it determines the actuators to which the decision shall be distributed based on its configuration management database.

- The *actuators* only consume decision events. An actuator checks if it is the recipient of decisions and enforces them if so. Like sensors, it is possible to develop actuators in any programming language.

B. Implementation and Tests

This section covers the implementation of the system and the results obtained after running tests on the scenario described in Section II-B. We implemented our scenario using the indoor positioning system in our laboratory where we installed a set of PoleStar[3] indoor positioning beacons. Due to size limit we cannot describe it in the article, however, a video showing our experimentation in our lab is available for watching[4].

The *message broker* is an Apache ActiveMQ server. We use Esper from EsperTech to develop the *situation manager*. The *command center* is built based on Balana, an XACMLv3 engine provided by WSO2. The policy editor is the eclipse plug-in that Axiomatics have developed to write ALFA policies. We also use the Balana API in all the components for generating/parsing XACMLv3 request and decision messages. Finally, we employed the java ActiveMQ SSL connector on Linux to allow each component to communicate with the broker; all components (Command center, Situation manager, Sensors and Actuators) are authenticated on the broker by an SSL certificate.

We have tested the first implementation of our scenario where laptops and smartphones send their GPS coordinates. In order to evaluate the performance of the broker, the situation manager, and the command center, we installed all the components on the same machine, which is a MacBook Pro with a 2.8GHz Intel Core i7 processor, 8Go 1600MHz DDR3 memory and a 256Go SSD drive. In this simulation environment, one java thread simulates the GPS sensors of the laptop and the smartphone for one user and a java thread plays the role of the actuators in laptops. Each simulation compels dynSMAUG to execute the complete process from context events sent by sensors to the reception of the decision by an actuator, i.e., reception of the GPS coordinates from laptop and smartphone, calculus of the situation, reception of the situation event by the command center, decision making process and reception of the decision by the actuator. We simulate 100 users. Each sensor sends a GPS coordinates update event every 10 seconds. Each simulation lasts 50 executions of the complete process for each user. Thus, we obtained the time of 5000 executions of the process.

The results obtained by the simulation show that the whole process took between 5ms and 91ms. 99% of all process executions were achieved in less than 19ms. The distribution

[3]http://www.polestar.eu/

[4]https://drive.google.com/file/d/0B-PPoWO4n0-XZWRsa21IWEVKNE0

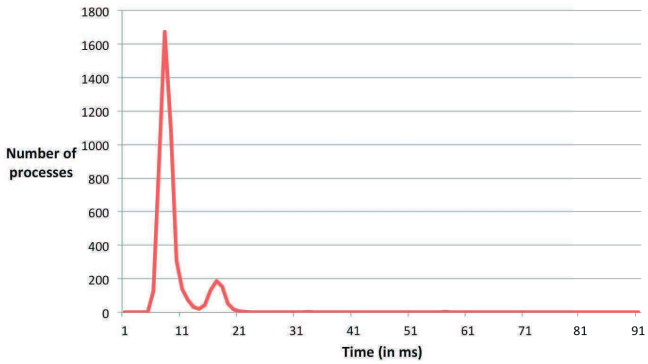

Fig. 7. Distribution of process execution times (total 5000 executions)

in Figure 7 shows a spike corresponding to 1674 process executions completed in 8ms. It represents around 30% of all the process executions. The other distinctive spike in the graph is at 17ms with 186 process executions. This test partially validates the viability of our system. However, these values should be weighted because of the small number of EPL and XACML rules.

V. Related works

Different research works have proposed to introduce the concept of context to allow dynamic permissions. For instance, Shebaro *et al.* [19] have included the context in the Android permissions management to support dynamic privileges. Here, context is a time interval and a physical location. The physical location is an abstract context information calculated based on GPS, cells and WIFI signals. This approach can not be easily extended to include more context information and policies are complex since they include all the technical details about what a physical location is. Bonati *et al.* [20] have extended the RBAC model to support reactive policies. Roles (and then permissions) can be activated/deactivated according to three context information: location, time and event that is "the description of other relevant measurable features of the context". This approach is limited to three context information and doesn't structure the dynamics of the world, contrary to the concept of situation. Finally, Son *et al.* [21] have analyzed deeper the notion of context and propose to express access control policies according to six axes of context: who, when, where, why, what and how. If this work seems close to what we call situation, they don't separate context from the policy and the dynamics is not structured.

As far as we know, only few researchers introduced the concept of situation in the security management process explicitly. Kim and Lim [22] have proposed *Situation-Aware-RBAC*, an extension of RBAC (Role Based Access Control) which dynamically grants roles to users thanks to a situation-aware matrix. This work mixes different context information for building a situation. However, the situation aware matrix is limited and only supports the specification of basic situations. Yau *et al.* [23] have presented a situation-aware access control approach that integrates situation-awareness capability and

RBAC. They propose a dedicated language for specifying situations. However, this language is limited comparing to Complex Event Processing languages and the specification of the situation is included inside the policy. Kayes *et al.* [24] have described an ontological framework for situation-aware access control. They proposed to formalize the concept of situation and context in an ontology. However, they have a different perspective about what a situation is. Unlike us, their approach focuses mainly on the purpose of access only. Finally, we presented a preliminary work in [25] that advocates the necessity of a situation manager. We applied our idea for implementing the Break-the-Glass mechanism [26]. However, our work was limited to authorization management and the deployment architecture was in a nascent state.

VI. Conclusion

We presented in this article, dynSMAUG, a framework that aims at making management of security more flexible and adaptive. The specificity of our approach consists in considering the concept of *situation* as the cornerstone of the security management. Situations allow to capture the dynamic constraints (time, location, workflows, etc.) and organize them into a stable and logical concept. We created a situation manager by applying Complex Event Processing (CEP) techniques. Since situation are calculated by the situation manager, the command center can consider them in its decision making process. We propose to specify situation oriented security policies in ALFA that simplifies XACMLv3 while keeping its expressiveness. Finally, we presented an infrastructure that supports both outsourcing and provisioning deployment modes. We performed different tests to evaluate our solution.

We plan to improve our calculus of situation by including the assessment of the situation assurance. Situations are deduced based on the context. However, the context information is intrinsically inaccurate and doesn't represent correctly the managed world. E.g, the accuracy of GPS coordinates is 5 meters, there is a delay up to 10 seconds with the indoor positioning system, etc. Therefore, we want to integrate the concept of *Quality of Context* [27] in the calculus of situations.

Acknowledgment

This work is part of the Box@PME project and has been funded by BpiFrance and Région Occitanie. Also, we would like to thank PoleStar for providing us with their indoor position technology.

References

[1] M. Harkins, *Managing Risk and Information Security: Protect to Enable*. Apress, 2012.

[2] D. Agrawal, K.-W. Lee, and J. Lobo, "Policy-based management of networked computing systems," *IEEE Communications Magazine*, vol. 43, no. 10, pp. 69–75, 2005.

[3] A. Westerinen, J. Strassner, M. Scherling, B. Quinn, S. Herzog, A. Huynh, M. Carlson, J. Perry, and S. Waldbusser, "Terminology for policy-based management, ietf rfc 3198," 2001.

[4] D. Chadwick, G. Zhao, S. Otenko, R. Laborde, L. Su, and T. A. Nguyen, "PERMIS: a modular authorization infrastructure," *Concurrency and Computation: Practice and Experience*, vol. 20, no. 11, pp. 1341–1357, 2008.

[5] OASIS, "eXtensible Access Control Markup Language (XACML) Version 3.0," Tech. Rep., 2013. [Online]. Available: http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-en.pdf

[6] L. Lymberopoulos, E. Lupu, and M. Sloman, "An adaptive policy-based framework for network services management," *Journal of Network and systems Management*, vol. 11, no. 3, pp. 277–303, 2003.

[7] V. K. Singh and R. Jain, *Situation Recognition Using Eventshop*. Springer, 2016.

[8] A. K. Dey, "Understanding and using context," *Personal and ubiquitous computing*, vol. 5, no. 1, pp. 4–7, 2001.

[9] M. R. Endsley, "Design and evaluation for situation awareness enhancement," in *Proceedings of the human factors and ergonomics society annual meeting*, vol. 32, no. 2. SAGE Publications, 1988, pp. 97–101.

[10] J. Barwise and J. Perry, *The situation underground*. Stanford University Press, 1980.

[11] A. Adi and O. Etzion, "Amit - the situation manager," *The VLDB Journal—The International Journal on Very Large Data Bases*, vol. 13, no. 2, pp. 177–203, 2004.

[12] D. Luckham, "The power of events: An introduction to complex event processing in distributed enterprise systems," in *International Workshop on Rules and Rule Markup Languages for the Semantic Web*. Springer, 2008, pp. 3–3.

[13] G. Cugola and A. Margara, "Processing flows of information: From data stream to complex event processing," *ACM Computing Surveys (CSUR)*, vol. 44, no. 3, p. 15, 2012.

[14] V. C. Hu, D. Ferraiolo, R. Kuhn, A. Schnitzer, K. Sandlin, R. Miller, and K. Scarfone, "Guide to Attribute Based Access COntrol (ABAC) Definition and Considerations," NIST, Tech. Rep. SP 800-162, 2016.

[15] R. Laborde, F. Barrère, and A. Benzekri, "Toward authorization as a service: a study of the xacml standard," in *Proceedings of the 16th Communications & Networking Symposium*. Society for Computer Simulation International, 2013, p. 9.

[16] R. Laborde, M. Kamel, F. Barrère, and A. Benzekri, "Pep = point to enhance particularly," in *Policies for Distributed Systems and Networks, 2008. POLICY 2008. IEEE Workshop on*. IEEE, 2008, pp. 93–96.

[17] A. Oglaza, R. Laborde, and P. Zaraté, "Authorization policies: Using decision support system for context-aware protection of user's private data," in *Trust, Security and Privacy in Computing and Communications (TrustCom), 2013 12th IEEE International Conference on*. IEEE, 2013, pp. 1639–1644.

[18] P. Giambiagi, S. K. Nair, and D. Brossard, "Abbreviated Language for Authorization Version 1.0," Mar. 2015. [Online]. Available: https://www.oasis-open.org/committees/download.php/55228/alfa-for-xacml-v1.0-wd01.doc

[19] B. Shebaro, O. Oluwatimi, and E. Bertino, "Context-based access control systems for mobile devices," *IEEE Transactions on Dependable and Secure Computing*, vol. 12, no. 2, pp. 150–163, 2015.

[20] P. Bonatti, C. Galdi, and D. Torres, "Event-driven rbac," *Journal of Computer Security*, vol. 23, no. 6, pp. 709–757, 2015.

[21] J. Son, J.-D. Kim, H.-S. Na, and D.-K. Baik, "Cbdac: context-based dynamic access control model using intuitive 5w1h for ubiquitous sensor network," *International Journal of Distributed Sensor Networks*, 2015.

[22] Y.-G. Kim and J. Lim, "Dynamic activation of role on rbac for ubiquitous applications," in *Convergence Information Technology, 2007. International Conference on*. IEEE, 2007, pp. 1148–1153.

[23] S. S. Yau, Y. Yao, and V. Banga, "Situation-aware access control for service-oriented autonomous decentralized systems," in *Autonomous Decentralized Systems, 2005. ISADS 2005. Proceedings*. IEEE, 2005, pp. 17–24.

[24] A. S. M. Kayes, J. Han, and A. Colman, "An ontological framework for situation-aware access control of software services," *Information Systems*, vol. 53, pp. 253–277, 2015.

[25] B. Kabbani, R. Laborde, F. Barrere, and A. Benzekri, "Specification and enforcement of dynamic authorization policies oriented by situations," in *New Technologies, Mobility and Security (NTMS), 2014 6th International Conference on*. IEEE, 2014, pp. 1–6.

[26] B. Kabbani, R. Laborde, F. Barrère, and A. Benzekri, "Managing Break-The-Glass using Situation-oriented authorizations," in *9ème Conférence sur la Sécurité des Architectures Réseaux et Systèmes d'Information-SAR-SSI 2014*, 2014.

[27] P. Marie, T. Desprats, S. Chabridon, M. Sibilla, and C. Taconet, "From ambient sensing to iot-based context computing: An open framework for end to end qoc management," *Sensors*, vol. 15, no. 6, pp. 14 180–14 206, 2015.