# A situation-driven framework for dynamic security management

Romain Laborde, Arnaud Oglaza, Ahmad Samer Wazan, François Barrère, Abdelmalek Benzekri

**Noname manuscript No.**
(will be inserted by the editor)

# A situation-driven framework for dynamic security management

**Romain Laborde · Arnaud Oglaza · Ahmad Samer Wazan · François Barrère · Abdelmalek Benzekri**

the date of receipt and acceptance should be inserted later

**Abstract** We present a dynamic security management framework where security policies are specified according to situations. Situation-based policies easily express complex dynamic security measures, are closer to business and simplify the policy life cycle management. Situations are specified using complex event processing techniques. The framework is supported by a modular event based infrastructure where a dedicated situation manager maintains active situations allowing the command center to take dynamic situation-based authorization and obligation decisions. The whole framework has been implemented and showed good performance by simulation. Finally, we detail two real experiments.

## 1 Introduction

The mission of IT security teams in organizations that consisted in *Protect to minimize risk* has shifted 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. Consequently, security has to evolve on an ongoing basis and the security management process should be flexible enough to quickly adapt the organization security to new usages/technologies and 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. But, 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.

This article extends [2] and presents *dynSMAUG*, our dynamic security management framework. The specificity of our approach consists in placing the concept of *situation* at the core of security management and writing policies herewith "when *situation* and *conditions* then *security actions*". On the one hand, situations capture the dynamic constraints (time, location, risk, etc.) and organize them into a stable and logical concept. Situation-based 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 enforced 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. We apply Complex Event Processing (CEP) techniques to calculate situations and security policies are specified using the Attribute Based Access Control paradigm in XACMLv3. Finally, dynSMAUG includes a modular deployment infrastructure to dynamically enforce situation-based security policies.

The rest of the article is structured as follows. Section 2 highlights the limits of security policy languages and architectures. In Section 3, we present the dynSMAUG framework. Section 4 deals with performance evaluation and two deployment experiments. Section 5 summarizes the related works. Finally, section 6 concludes and provides some perspectives.

R. Laborde · A. Oglaza · S. Wazan · F. Barrère · A. Benzekri
University Paul Sabatier
118 Route de Narbonne, F-31062 Toulouse CEDEX 9, France
E-mail: Romain.Laborde@irit.fr

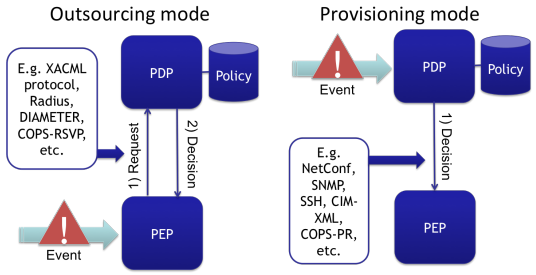


**Figure 1** Policy-Based management modes

## 2 Description of the problem

We first summarize the different existing approaches for managing security. Then, we present a scenario showing the limitations of these approaches.

### 2.1 Policy-based security management approaches

Policy based management (PBM) is a recognized approach for managing complex systems that externalizes the management logic out of the managed systems [3]. The architecture supporting this approach consists of a policy decision point (PDP) that interprets the policy and takes decisions 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 deployment exist [4] which differ on 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 latter 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 'If *condition* then *permit/deny*'. PERMIS [5] is an example of such systems. The *provisioning mode* is more suitable for deploying security configurations [6]. In this mode, the PDP can receive an event from any sensor to trigger the decision-making process. The PDP then transmits its decision to the relevant PEP(s). 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 [7] is an example of such systems.

### 2.2 Scenario

Let us consider a company working with sensitive 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, 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 sensitive data at risk of being disclosed, the company decided to automate this task.

The enterprise proposes to improve the initial *login/password* policy by locating both users and computers in order to automatically lock the sessions when the owner is moving far from his/her associated device (e.g., the user exits the room). 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/her device, the access must be refused and a notification message must be sent to the owner via an alternative channel. In the rest of the article, this rule is called *DENY-AUTHZ-RULE*. Finally, connection is allowed only if the login/password is valid and the owner is in front of his/her device (rule *PERMIT-AUTHZ-RULE*).

As a first implementation, the company decided to collect the locations of computers based on their GPS chipset. After agreeing with the employees upon privacy issues, the company also chooses to use the GPS system embedded in the employees' smartphones. Using this tracking solution, *REACTIVE-RULE*, *DENY-AUTHZ-RULE* and *PERMIT-AUTHZ-RULE* can be rephrased:

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

**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 or equal 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., they are in two different rooms). Thus, this mechanism poorly

implements the ISSP. To improve it, the company decided to use an indoor positioning system that combines GPS, Wi-Fi, Bluetooth Low Energy, and motion sensors. This system is more precise than GPS because it is able to determine the room where computers and users stand. However, introducing this new technology requires to modify the three policy rules to become:

**REACTIVE-RULEv2**
*On* (the owner and the computer are no more in the same room) *then* lock the session of the computer

**DENY-AUTHZ-RULEv2**
*if* (the connection to a computer is authenticated) *and* (the owner and the computer are not in the same room) *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 and the computer are in the same room) *then* permit the connection

This scenario highlights two requirements:

*REQ1* : The security policy shall be independent from technology. In our example, each ISSP rule has been refined into different rules, one for each position technology. If new security rules are added each time a new security related-technology is introduced, the security policy will become unmanageable.

*REQ2* : The security management system shall support both outsourcing and provisioning modes in a unified way.

## 3 The dynSMAUG framework

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

### 3.1 Definition of situation

Although the word *situation* is commonly employed, it has many definitions and is often interchanged with the word *context*. We present in this section our interpretation of these two concepts.

Dey [8] has proposed one of the most popular definitions of *context*: "*context is any information that can be used to characterize the situation of an entity.*" Hence, *situation* is something more abstract than *context*.

Furthermore, Barwise and Perry [9] provided a first definition of 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.* This definition highlights that situation is about making relations between context information.

Endsley [10] supplemented this work in her definition of situational awareness: [It is] *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 another characteristic of situation. Situation is about understanding the context and being able to project in the future. As a consequence, unlike context, a situation can be qualified as "good" or "bad" based on properties of the current situation as well as the possible future situations. Understanding what is happening and what could happen in the future improves decision making.

Therefore, from a security management perspective, we consider that the *context* is every instantaneous, detectable, and relevant security related information of the managed environment collected by sensors such as monitoring systems, intrusion detection systems, configuration management databases, etc. A *situation* is specific time frame during which the result of some computed relationships between part of the collected security events is stable. A situation focuses on specific entities of interest to protect, which we call *situation target*, and a particular *security concern*.

Our scenario in section 2.2 incorporates two situations that characterize a geographical relation between a computer and its owner (Figure 2). Situation *close* (referring to "the computer is close to its owner") can be

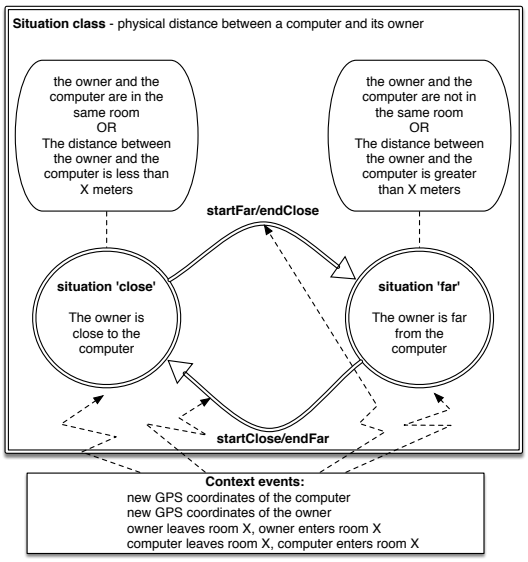


**Figure 2** Situations in our scenario

qualified as rather safe unlike situation *far* (referring to "it is far from its owner") that is more risky. In addition, these situations are closely related since they both focus on the same *security concern* (here the risk associated to the computer). We say that the situations related to the same *situation target* and the same *security concern* are in the same *situation class*. A situation class forms a weakly connected graph where vertices are the situations of the class and edges are context events impacting the stability of the situations (Figure 2). Situations of the same class are mutually exclusive, i.e., an entity can only be in a single situation of a given class at the same time. However, any entity can find itself in several situations in parallel provided all the situations belong to different classes.

Describing the dynamics of the context as a graph of situations facilitates the specification of security policies because it is closer to business needs. For instance, the ISSP is directly translated to:

- When the situation is moving from *close* to *far*, the session of the computer should be locked automatically (REACTIVE-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 *close*, the user can connect to the computer (PERMIT-AUTHZ-RULE).

However, computing situations *close* and *far* depends on the available sensors and context data.

### 3.2 Specification and calculus of situations

A situation, being a particular time frame of interest, 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 involving multiple entities and multiple conditions, the beginning and the end of a situation cannot be simple events captured by a single 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* and *far* should be computed based on events related to the position of the user and the computer. 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 entities is greater than X meters. Alternatively, when events report that entities are entering or leaving rooms, the same situation starts by determining that the owner and the computer are no more located in the same room.

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 specifying 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. We choose Esper[1], the open source event processing implementation maintained by Espertech. Esper offers a stream-oriented language for specifying complex event patterns, called Event Processing Language (EPL), that is an extension of SQL for processing events (e.g., windows definition and interaction, timed-data arithmetic definition, etc.)

In our scenario, situation *far* begins when situation *close* ends and vice versa (Figure 2). Thus, identifying situations *far* and *close* consists in expressing in EPL the two complex events *'startFar/endClose'* and *'startClose/endFar'*. Figure 3 is the EPL description of *startFar* when the location of users and computers is determined by their GPS positions. In our example, GPS sensors send the GPS position every 10 seconds. This EPL rule states that complex event *startFar* means *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 current situation is close (lines 13-17).*

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

```
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  fr.irit.GPS.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  current-situation.value = "close"
```

**Figure 3** Situation "far" using GPS

```
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 lockSession {
8        Attribute.recipientTarget=
9           Situation_laptop_owner.owner
10       Attribute.recipientType="laptop"}
11   }
12 }
13 rule deny_authz_rule {
14   deny
15   target clause
16     Situation_laptop_owner.value == "far"
17     and Action.name=="authenticated-connection"
18   on deny{
19     obligation notifyUser {
20       Attribute.recipientTarget=
21          Situation_laptop_owner.owner
22       Attribute.recipientType="smartphone" }
23   }
24 }
25 rule permit_authz_rule {
26   permit
27   target clause
28     Situation_laptop_owner.value == "close"
29     and Action.name == "authenticated-connection"
30 }
```

**Figure 4** Our example ISSP Policy in ALFA

The strategy for specifying the beginning and the end of situations depends on the input events and the characteristics of the sensors. The specification of complex event *startFar* using the indoor positioning system is explained later in section 4.2.

### 3.3 Specification of situation-based security policies

In our approach, situations are specified outside the security policy. Therefore, the security policy can just refer to them. Hence, 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 2.2, 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* then *authorization decision and/or obligation(s)*

We propose to express security policies in XACMLv3 [13], 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 terms of constraints on security attributes; attributes being any characteristics of entities. Attributes can be grouped into predefined categories (subject, action, resource) or user-defined 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]. However, XACML is a verbose XML language which makes difficult to write or read security policies [16]. Abbreviated Language for Authorization (ALFA) [17] has overcome this issue by providing the means to express XACMLv3 policies in more compact and human-readable forms.

Figure 4 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 2.2, which limits the errors when translating high level security policies into a target security policy language. The rule called *reactive_rule* stands for: 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). Attributes *recipientTarget* and *recipientType* in the obligation (lines 8-10) indicate where to enforce the obligation. If they are not filled, the obligation is returned to the management entity indicated in the request (outsourcing mode). The second rule *deny_authz_rule* 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 as an 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_laptop_owner is close* (line 30) and *the connection is authenticated* (line 31).

### 3.4 The deployment infrastructure

The architecture of dynSMAUG aims at allowing the deployment of security policies in both outsourcing and provisioning modes (Figure 5). 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 plays the role sensor. Secondly, we consider every message as event to unify the interactions between the sensors/actuators and the PDP. Hence, the protocol in the outsourcing mode consists of 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).

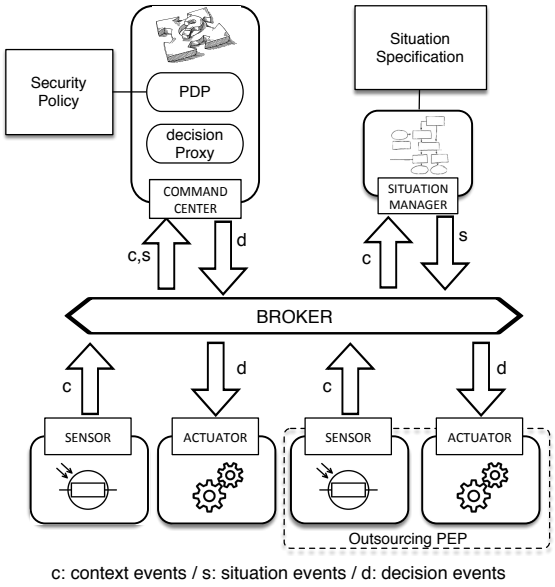


**Figure 5** Architecture of dynSMAUG

The actors of our deployment architecture are shown in Figure 5:

- 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 kinds of topics: the context events, the situation events and the decision events. It is also responsible for controlling the access to the dynSMAUG infrastructure (each dynSMAUG actor is authenticated by an X.509 certificate).
- The *sensors* produce context events (noted $c$ in Figure 5). A context event can be the GPS coordinates, the user exiting a room, or an attempt to access a protected resource. Each 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 CEP engine that calculates situations according to a situation specification as explained in section 3.2. It consumes context events and produces situation events (noted $s$ in Figure 5). Situation events have the same format as context events and are also carried in XACMLv3 format. Each time a new situation is calculated, the situation manager creates two situation events: the beginning of the new situation and 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 5). As explained in section 3.3, we specify security policies in XACMLv3. However, XACML PDPs only implement the outsourcing mode. Therefore, the command center includes also a decision proxy for allowing the command center to operate both outsourcing and provisioning modes. Hence, the PDP acts in compliance with XACMLv3. When the decision proxy receives the decision from the PDP, it publishes the decision to the correct topic(s) based on attributes *recipientTarget* and *recipientType* (see Figure 4) to distribute it the relevant actuators.
- 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.

## 4 Implementation and deployment examples

We have implemented the whole dynSMAUG architecture and evaluated its performance. We also deployed it during two real experiments. The *message broker* is an Apache ActiveMQ server. All the components (Command center, Situation manager, Sensors and Actuators) are authenticated on the broker using SSL certificates. We use Esper to develop the *situation manager*. We built the *command center* based on Balana[2], an XACMLv3 engine provided by WSO2. The policy editor is the ALFA eclipse plug-in developed by Axiomatics. We also use the Balana API in all the components for generating and parsing XACMLv3 messages.

### 4.1 Performance evaluation

According to EsperTech, *Esper exceeds over 500 000 event/s on a dual CPU 2GHz Intel, with engine latency below 3µs average with 1000 EPL statements.* This allows the situation manager to integrate operational security environments and calculate complex situations.

We also evaluated the performance of our system on the first implementation of our scenario (i.e., position based on GPS coordinates) by simulating GPS sensors and actuators by java threads. We installed all the components on the same machine (a MacBook Pro with a

[2] `https://github.com/wso2/balana`

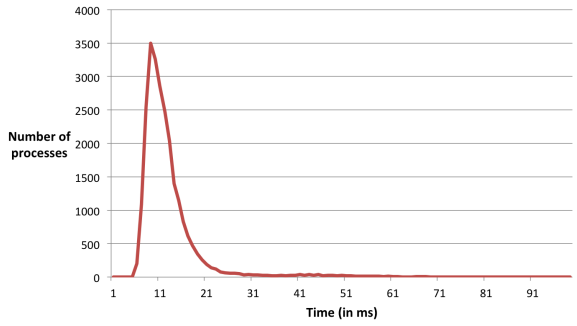


**Figure 6** Distribution of execution processes with 500 users (25000 processes)

2.8GHz Intel Core i7 processor, 8Go 1600MHz DDR3 memory). 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 simulated 500 users and each sensor send GPS coordinates every 10 seconds. Each simulation lasts 50 executions of the complete process for each user resulting in 25000 executions of the process. The distribution of time (Figure 6) shows that 90% of all process executions were achieved in less than 18ms and 98% in less than 55 ms. The worst execution time was 195 ms due to the broker (96% of the total time).

## 4.2 Dynamic security based on indoor location

We implemented the scenario of section 2.2 using the indoor positioning system in a real environment (Figure 7). We deployed an indoor position solution lent by the PoleStar company[3] in our laboratory covering the lobby and three rooms where we installed a set of Bluetooth Low Energy beacons that broadcast a signal conveying their ID. After a learning phase that scans the signal power received in different places of every room, Android and iOS devices are able to deduce their position inside a building through a proprietary API. A video of the experimentation in our lab is available for watching (`https://www.youtube.com/watch?v=wNnL_oqAUK0`).

In this experiment, we embedded the command center and the situation manager in a Raspberry Pi 3, the smartphone device is a Google Nexus 5X and the laptop is an Ubuntu Linux. We developed on the Android smartphone an actuator that can display notifications and a positioning sensor in Java that publishes context events when the smartphone moves to another room

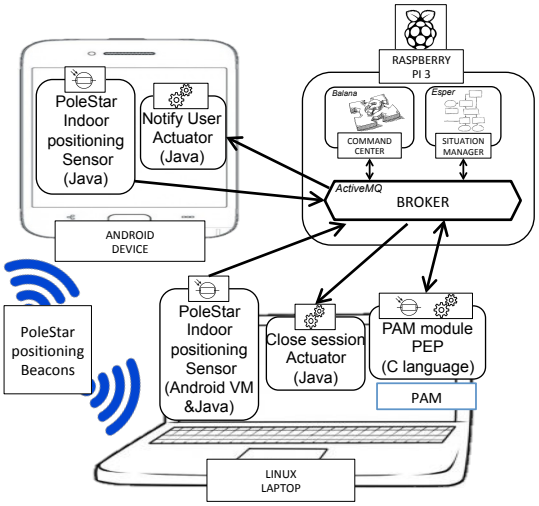


**Figure 7** Indoor positioning system implementation

(the PoleStar indoor positioning API triggers events of the form “entering in room X"). On the laptop, we developed an outsourcing PEP as a Linux Pluggable Authentication Module (PAM) that extends the native Linux authentication process. Each time a user enters a username/password, the PAM module asks the command center for a decision. If the command center denies the access, the PAM module blocks the session login process. We also provided an actuator that can lock the current session. Finally, the indoor positioning API being only available for Android and iOS, the position sensor on the laptops runs under Android-X86[4].

The command center is configured with the same policy as with the GPS solution (Figure 4) proving that our approach makes the policy independent from the users tracking technology. However, the situation manager uses a different situation specification due to the specificities of the indoor positioning events. Indeed, indoor position sensors only trigger events when smartphones or computers are entering in a new room unlike GPS events that are sent every 10 seconds. As a consequence, the complex event *startFar* appears either when the user or the computer has moved to another room while both were initially in the same room. In addition, other technical features have to be considered. The indoor positioning system deployed in our lab takes between 0 to 10 seconds to calculate the position of a device. This constraint of 10 seconds is specific to our lab experimentation. Better performance can be achieved with more beacons. Therefore, there may be a delay of 10 seconds between the time a device actually enters a room and the time the context event report-

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

[4] `http://www.android-x86.org/`

ing this change is actually generated. Thus, situation *far* starts when there is event *computer enters room X* (resp. *smartphone enters room X*) while the current situation is *close*, and no event *smartphone enters room X* (resp. *computer enters room X*) is triggered within 10 seconds. To translate this statement in EPL, we specify event *startFar* in two steps (Figure 8). First, we compute two complex events: *i*) *MayBeFarEvent* meaning 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 will be re-injected within the CEP system to compute the beginning of situation *far* by when an event *MayBeFarEvent* has been triggered

```
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))]
```

**Figure 8** Situation "far" using indoor positioning

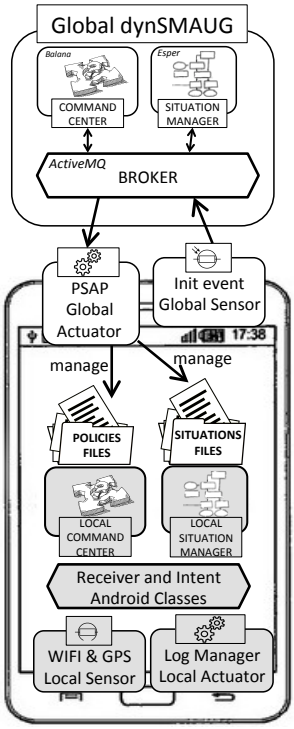


**Figure 9** Smart node architecture

(line 38) and after waiting 10 seconds (line 39) no event *MayBeCloseEvent* appears for the same owner (line 40).

### 4.3 Locally-decided dynamic security measures

The second experiment is related to collecting logs by a Security Operations Center (SOC) where mobile devices should upload log files to a dedicated SOC server. However, logs containing sensitive information should be sent only when the mobile device is in a secure environment. In our case, being in a secure environment is equivalent to being connected to specific WIFI networks inside predefined geographical areas.

We developed a decentralized management architecture because, the local context being captured by the mobile device itself, making situation-based decisions at the device is far more efficient. Hence, we integrated the whole dynSMAUG architecture within the mobile device (the grey elements in Figure 9) that we call *smart node*. We developed the local command center using Balana and the local situation manager thanks to ASPER[5]. Finally, a local broker is implemented using the native Android messaging mechanism (BroadcastReceiver and Intent classes). Thus, the smart node has its own local policy (Figure 11) and computes its own local situations (Figure 10). Hence, smart nodes can apply dynamic security measures autonomously.

[5] `https://github.com/mobile-event-processing/Asper`

```
startSecureEnvt = select * from
 LocationEvent.std:lastevent() as location,
 WifiEvent.std:lastevent() as wifi,
 Active_Situation.std:lastevent() as current-situation
where
 current-situation.value = "unsecure" and
 fr.irit.SOC.isInBuilding(location, MYBUILDING) and
 wifi.name = "secure_wifi"
```

**Figure 10** Local situation "secure-envt" in EPL

```
rule permit_send_logs {
  permit
  target clause
    Situation_environment.value == "secure-envt"
    and Situation_environment.state=="begin"
  on permit{
    obligation uploadLogs{
      Attribute.recipientTarget="self"
      Attribute.recipientType="logManager"
      Attribute.SOCServerAddress="192.168.1.10"
      Attribute.SOCServerPort="12345"}
}

rule deny_send_logs {
  permit
  target clause
    Situation_environment.value == "unsecure-envt"
    and Situation_environment.state=="begin"
  on permit{
    obligation storeLogs{
      Attribute.recipientTarget="self"
      Attribute.recipientType="logManager"}
}
```

**Figure 11** The local secure log upload policy in ALFA

```
rule log_smart_node_init_policy {
  permit
  target clause
    Init.id="init-device"
    Init.role="log-smart-node"
  on permit{
    obligation loadPolicyAndSituation{
      Attribute.policyID="log-managementv1"
      Attribute.policy=
        "PD94bWwgdm....BvbGljeT4K"
      Attribute.situationID="log-managementv1"
      Attribute.situationSpec=
        "dW5zZWN1cm....NC4wKSkK"}
}
```

**Figure 12** Sample of the global policy in ALFA

*Smart nodes* can be dynamically managed by the global command center like any device. To allow such feature, a smart node must include an actuator called Policy and Situation Administration Point (PSAP) that is responsible for managing the local policies and local situation specifications. The first time the smart node connects to the global broker, a specific sensor on the smart node sends an *init* message signalling it has no policy nor situation specification. The global command center has specific rules for managing smart nodes like the rule in Figure 12 stating that when a smart node collecting logs for the SOC sends an init event then the PSAP of the smart node must load a given policy and situation specification (the policy and the situation specification are encoded in base64 for technical reasons). When the PSAP of the smart node receives this obligation, the local policy (resp. local situation specification) is loaded in the local command center (resp. the situation manager). It is worth noting that our system allows the global command center to dynamically change the local policy/situation specification of any smart nodes according to any global situation changing.

## 5 Related works

Different research works have proposed to introduce the concept of context to allow dynamic permissions. For instance, Bonati *et al.* [18] have included time and physical location inside policies to support dynamic privileges. However, this work is limited only to few context features and doesn't provide context abstraction such as our concept of situation. Son *et al.* [19] proposed to express access control policies according to six axes of context: who, when, where, why, what and how. Nonetheless, they don't separate context from the policy and the dynamics is not structured.

Kim and Lim [20] have proposed *Situation-Aware-RBAC* which dynamically grants roles to users according to a situation-aware matrix. However, the situation-aware matrix only supports the specification of basic situations. Yau *et al.* [21] integrated situation-awareness capability and RBAC. Likewise, the situation specification language is limited and the situation specification is included inside the policy. Kayes *et al.* [22] have described an ontological framework for situation-aware access control. However, their approach focuses mainly on the purpose of access only. Finally, we presented a preliminary work in [23] that advocates the necessity of a situation manager and we implemented the Break-the-Glass mechanism [24]. However, our work was limited to the outsourcing deployment mode only.

## 6 Conclusion

We presented dynSMAUG, a framework that makes security management more flexible and adaptive by placing the concept of *situation* at the cornerstone of the security management. Situations allow capturing the dynamic constraints (time, location, workflows, etc.) using CEP techniques and organize them into a stable

and logical concept. Delegating the calculation of situations to a dedicated situation manager simplifies the security policies by making them closer to business needs. Our deployment architecture supports both outsourcing and provisioning deployment modes in a unified approach and can also be easily extended by writing actuators/sensors in most of existing programming languages. Finally, we demonstrated that the same framework can be applied at different levels from the whole organizations' IT system to devices.

Our future work will focus at improving the calculus of situations. First, we'll include the assessment of the situation assurance by integrating the concept of *Quality of Context* [25]. We also want to explore the interactions between global situations calculated at the organization level and local situations computed at the smart nodes level. For more long-term, we plan to create a complete situation-based security engineering methodology covering the whole security process from adaptive security requirements to adaptive security enforcement. Especially, we envision to build a formal situation-based access control model.

## Acknowledgment

This work, part of the Box@PME project, was funded by BpiFrance and Région Occitanie. We would like to thank PoleStar for their indoor position technology.

## References

1. M. Harkins, *Managing Risk and Information Security: Protect to Enable.* Apress, 2012.
2. R. Laborde, A. Oglaza, F. Barrère, and A. Benzekri, "dynsmaug: A dynamic security management framework driven by situations," in *Cyber Security in Networking Conference (CSNet), 2017 1st.* IEEE, 2017, pp. 1–8.
3. 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.
4. 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.
5. 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.
6. F. Barrère, A. Benzekri, F. Grasset, and R. Laborde, "A multi-domain security policy distribution architecture for dynamic IP based VPN management," in *3rd International Workshop on Policies for Distributed Systems and Networks (POLICY 2002)*, 2002, pp. 224–227.
7. 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.
8. A. K. Dey, "Understanding and using context," *Personal and ubiquitous computing*, vol. 5, no. 1, pp. 4–7, 2001.
9. J. Barwise and J. Perry, *The situation underground.* Stanford University Press, 1980.
10. 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.
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 *Workshop on Rules and Rule Markup Languages for the Semantic Web.* Springer, 2008, p. 3.
13. 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
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.* SCS, 2013, p. 9.
16. 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), International Conference on.* IEEE, 2013, pp. 1639–1644.
17. 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
18. P. Bonatti, C. Galdi, and D. Torres, "Event-driven rbac," *Journal of Computer Security*, vol. 23, no. 6, pp. 709–757, 2015.
19. 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.
20. 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.
21. 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.
22. 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.
23. 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.
24. 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.
25. 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.