Context-based Risk-Adaptive Security Model and Run-time Conflict Analysis

Mahsa Teimourikia, Guido Marilli, and Mariagrazia Fugini

Politecnico di Milano,
Via Ponzio 34/35, 20133 Milan, Italy
mahsa.teimourikia@polimi.it,guido.marilli@mail.polimi.it
mariagrazia.fugini@polimi.it

Abstract. In dynamic and risk-prone environments, security rules should be flexible enough to permit the treatment of risks, and to manage privileges on physical and virtual resources for various authorized users based on the situation at hand. For this purpose, we define safety-centric contexts based on risk description that is provided by the Safety Management System (SMS). This paper presents a risk-adaptive Access Control System (ACS) that adopts hierarchies of contexts and Access Control Domains (ACDs) to make adaptations to risks related to safety at different levels of criticality. Since various risks may arise simultaneously, two or more ACDs might be applicable at the same time incorporating various security rules which might lead to conflicts. Therefore, an approach to dynamically detect and resolve conflicts is essential. In this work, we propose a run-time conflict analysis and resolution algorithm based on XEngine and we illustrate its usage with the proposed risk-adaptive ACS.

Keywords: attribute-based access control; security; xacml; security policy conflict analysis; context-awareness; safety management.

1 Introduction

Today, access control paradigms are moving from traditional models such as Role-Based Access Control (RBAC) to Attribute-Based Access Control (ABAC) [6], which offers a fine-grained access control over resources by considering relevant attributes for users, resources and the environment, and hence, it enables more expressive security policies. On one hand, access control models are also being applied on physical and virtual resources [4, 3, 10], specially in smart work environments where the "things" (i.e., machinery and tools) are interconnected to form the Internet of Things (IoT). On the other hand, in a risk-prone environment such as construction and process industries, security policies should be flexible enough to permit the treatment of risks when necessary, and to manage privileges on physical and virtual resources for various authorized users based on the situation at hand.

Context-awareness in security is concerned with adaptation of security rules at run time to the situation at hand. In smart environments, adopting Internet

of Things (IoT), various monitoring data is available to recognize the situational factors, facilitating incorporation of context-awareness into access control of physical resources. In these environments, security rules should be managed adaptively based on the events that arise on the fly, such as the events that indicate potential risks to people's safety [3].

Previously, we presented a risk-adaptive ACS based on ABAC where we adopted Access Control Domains (ACDs) that include set of security rules defined for specific risk situations [3]. In the proposed approach, we assumed dealing with only one risk at a time to avoid unpredictable conflicts that may arise if more than one ACD be applicable simultaneously. Considering dynamic combination of ACDs, conflicts between security rules may arise which should be tackled at run-time. We adopted XACML that enables usage of combination algorithms at rule and policy levels to avoid conflicts. However, in our case this is not enough and still conflicts may arise because of the dynamic changes of the combinations of security rules.

Furthermore, in order to avoid repetition of the security rules in the ACDs there is a need to consider hierarchies of ACDs where child ACDs inherit the security rules defined in their parents. In addition, it should be taken into consideration that when there is a case of an emergency or crisis, there is not enough time to reason about the situation at hand and an approach should be considered for managing the emergency situations at run-time.

This paper, presents our approach to resolve mentioned problems in our risk-adaptive ACS. We define safety-centric contexts based on the risk description provided by the Safety Management System. We also propose an approach for calculating the level of criticality of the contexts to form hierarchies of contexts, each related to an ACD that deals with risks with different types and at different levels of criticality. Since, various risks can arise simultaneously, two or more ACDs would be applicable at a time causing unpredictable conflicts on run-time. To overcome this problem, we propose a run-time conflict analysis and resolution algorithm based on XEngine which dynamically resolves conflicts and adequately merges security rules that are simultaneously applicable. Furthermore, to manage the emergency situations that are time-critical, we consider a "break-glass" ACD that is activated when one or more of the context's criticality level is "emergency".

The paper is organized as follows: Section 2 reviews the state of the art. In Section 3 we introduce the preliminary concepts. Section 4 illustrates a scenario that puts in evidence a typical case of policy conflict. Section 5 shows the architecture of the Risk-Adaptive Access Control System (ACS). Section 6 presents our solution for conflict analysis and its resolution at run-time based on the proposed architectural framework. Section 7 gives some details about how we implemented our solution. And finally, Section 8 contains concluding remarks and ideas for future work.

2 Related work

Access Control Systems (ACSs) are the first line of defense in the overall security of a system. ACS applied to physical and virtual resources mainly protects the access points. Traditional ACSs such as Mandatory Access Control (MAC), Discretionary Access Control (DAC) and Role-Based Access Control (RBAC), mainly focus on defining user rights precisely to avoid any violations of the defined security rules [7]. However, traditional access control models demonstrate limited capabilities for adaptations to dynamic changes because of considering static security rules that determine the authorization decisions [7].

Risk-Adaptive Access Control is an emerging topic in the current research [3, 13, 1] which mainly concerns with the balancing the risk of granting or denying access to resources. While the research has mostly focused on security risks management in access control models [2, 1], there are limited works on considering adaptations based on safety risks.

In [13], authors propose a criticality-aware ACS based on RBAC, where, according to the critical state of the environment, privileges of users can be dynamically altered by changing the user roles and the Access Control Lists (ACL) associated to the resources. Due to the adoption of RBAC, in the dynamic authorization process, they do not consider relevant attributes such as the location of the person or physical devices but they limit their proposal to the clearance of users. Moreover, dynamic authorizations, namely authorizations that are varied at run time according to what happens in the environment must be kept compliant with the overall security objectives of organizations, ensuring that there is always a balance between security and safety based on defined protocols of the organization.

ABAC has a potential to enable fine-grained access control in IoT applications because of its ability in accommodating changes to various attributes of users and resources to promote fine-grained and dynamic AC [6]. In [3], we described our approach, based on ABAC, to manage the dynamic changes to the security-related attributes of users and resources to dynamically authorize their privileges based on risks that are detected in the environment. In this work, we extend [3] to introduce the use of hierarchies of contexts and ACDs. Contexts in our view are safety-centric and represent a situation based on risks detected in the environment and their level of criticality. With the valuable data gathered in IoT environments, risks concerning the safety of people at various levels of criticality can be identified [5]. To manage risks, different approaches employ break-glass policies in ACS [11]. However, more flexible and fine-grained ACS can be applied having various, yet finite and manageable sets of security policies (defined as access control domains) for different contexts.

However, during dynamic adaptations of security policies, conflicts may arise. Conflict detection and resolution has been an interesting topic that attracted great amount of research during current years [12, 9, 14]. In [12], authors translate restrictions in ACLs via OpenLDAP and offer a GUI that lets the administrator manually resolve the highlighted conflicts. However, this approach is not feasible where it may be necessary to combine undefined numbers of ACDs at run time,

because it requires long computation times that are incompatible with run time needs.

In this paper, we propose an approach similar to XEngine [8], which is an efficient XACML policy evaluation engine and therefore, it can be adopted at run-time with acceptable performance. XEngine enables management of multi-valued attributes and requests and resolves the issue of merging policies with different combining algorithms by normalizing them to "First-applicable". However, since this approach only applies on policies with the same target, we extend it in order to apply it on ACDs with different targets (a target of an ACD in our case is simply identified by the context).

3 Preliminary Definitions

In this section, we define the preliminary concepts that will be used throughout this paper. Risks related to safety in industrial environments refer to the threats that might endanger the health or life of the workers, which we simply refer to as "risks" from now on. As a policy language we adopt XACML, which its components are described in Table 1.

Table 1: XACML Components

XACML Component	**Description**
Policy Set	It's a set of policies, characterized by a target and a combining algorithm.
Policy	It's a set of rules, that apply to a certain target. Its result is computed basing on the chosen combining algorithm.
Rule	It's contained in a policy and is composed by a target, condition and an effect.
Target	Describes a set (or range) of values for the various categories' attributes, under which the policy/rule is applicable.
Condition	It's an expression in a rule that evaluates to true or false and along with the target determines the effect of the rule/policy.
Effect	It's the outcome of a rule/policy. The possible allowed values are usually *permit* and *deny*.
Policy Combining Algorithm	It's the procedure according to which the results of the policies in the policy set are combined.
Rule Combining Algorithm	It's the procedure according to which the results of the rules of a policy are combined.
Attribute	Characteristic of a subject, resource, action or environment. Each category usually has a set of attributes.

Table 2 lists the four main rule/policy combining algorithms in XACML, namely: deny overrides; permit overrides; first applicable; only one applicable.

Table 2: Combining Algorithms

Algorithm	Description
Deny-overrides	If any evaluation returns deny, then the result must be deny, even if other evaluations have returned permit.
Permit-overrides	If any evaluation returns permit, then the result must be permit, even if other evaluations have returned deny.
First applicable	Rules are evaluated in their listing order.
Only-one-applicable	For all of policies in the policy set, if no policy applies, then the result is NotApplicable. If more than one policy applies, then the result is Indeterminate. If only one policy applies, then the result is the result of evaluating that policy.

Moreover, Table 3, summarizes the basic definitions regarding the proposed ACS. These include: Subject S, Object O, Environment EN, Privilege P, Access Control Domain ACD, Security Rule SR, Monitoring Device MD, Hazard H, Risk R, and Consequence .

Table 3: Definitions & Notations

Notation	Definition
S	Finite set of entities both needing authorization to access resources (e.g., safety teams) and needing protection against risks (e.g., workers).
O	Finite set of physical resources or "things" (objects), e.g., tools, machinery, devices, that subjects can access or act on.
EN	Finite set of environment sections.
P	Finite set of privileges that are actions which subjects can perform on objects.
ACD	Finite set of access control domains that contain security rules designed for different contexts.
SR	Finite set of security rules.
MD	Finite set of monitoring devices that sense data from S, O, and/or EN, e.g., sensors, cameras, wearable sensors, etc.
H	Finite set of hazards acknowledged via events in the environment; hazards might turn into risks.
R	Finite set of risks identified in EN and endowed with attributes such as $Type$, $Probability$, $Source$, $Location$, and $Consequence$.
C	Finite set of consequences which originate from each $r_i \in R$, endowed with attributes such as: $Type$, $Intensity$, $Probability$.

ABAC is the basis of the the ACS, where S, O, and EN and their attributes, SA, OA, and ENA respectively, are evaluated by the ACS for a fine-grained authorization, considering the applied SR. We distinguish two types of attributes ($ATTR = SA \cup OA \cup ENA$): 1) security related $ATTR_{sec}$, (e.g., *security level*, *role*) that are defined by the security administrator; and 2) context related attributes $ATTR_{context}$, like *location*,

and *time*, that are dynamically set when the relevant data is received from *MD*. *EN* is considered to be an Smart Work Environment (SWE), where, $ATTR_{context}$ defines the global safety-centric context that identifies the risk type and its intensity that affects the whole or parts of the environment including the subjects and objects inside it. To adopt XACML, SR is mapped to the Rule component, ACD is mapped to the Policy component and the applicable ACDs are considered as the Policy Set in XACML.

4 A Motivating Scenario

In this section, we introduce a motivating scenario to illustrate a use-case in which conflicts may arise. In our approach under "safe" conditions an ACD defined for the safe context applies. If we enter into a risk state with a particular intensity, the related ACD applies that contain security rules previously designed based on the organizations policies and protocols for management of that specific risk which usually relax some otherwise restricted security rules. Considering the possibility of dealing with more than one risk simultaneously, security rules should allow management of all the risks that are present. Therefore, when there are conflicts, rules permitting the execution of preventive or corrective strategies for risk management should prevail. While, in some cases we might want to restrict some permissions for safety reasons, e.g., a machinery is detected to be faulty and we want to restrict access to that machinery to avoid eventual risks. Hence, we cannot always have the assumption of Permit-Overrides when combining several ACDs.

In what follows we make an example for clarifying the issue. Assuming to have following ACDs, defined for two different risk situations:

$$
\begin{aligned}
&acd_1 : \textbf{Context} = ShortCircuit \\
&\quad ru_1 : \textbf{IF}\ \{req.o.Type == "FireSprinkler" \wedge req.p == "TurnOn"\} \\
&\quad \textbf{THEN}\ \{effect == Deny\}
\end{aligned}
$$

$ru_1 \in acd_1$ indicates that if anyone tries to activate the fire sprinkler system when there is a risk of electrical short circuit, the effect must be deny (water on a electrical short circuit may cause electric shock).

$$
\begin{aligned}
&acd_2 : \textbf{Context} = Fire \\
&\quad ru_1 : \textbf{IF}\ \{req.s.ActiveRole == "RiskManager" \wedge req.o.Type == "FireSprinkler" \\
&\quad \wedge req.p == "TurnOn"\} \\
&\quad \textbf{THEN}\ \{effect == Permit\}
\end{aligned}
$$

While, $ru_1 \in acd_2$ indicates that in case of fire, the risk manager should to be permitted to turn on the fire sprinkler system if necessary.

Therefore, If *electrical short circuit* and *fire* risks are arise simultaneously, ACS have to consider both security rules at the same time. In this case, if the fire sprinkler system starts, the short circuit could intensify the fire and cause electric shock so the proper PolicySet combining rule (reminding that XACML PolicySet indicated the applicable ACDs in our case) should be deny-overrides. In conclusion, the hypothesis that the in case of multiple risks the most permissive security rule should prevail is not always correct.

Another interesting case happens if the PolicySet combining rule is First-applicable. Considering the particular nature of our system, the order of applicability of the ACDs is not predictable and therefore, it is possible that the ACDs will be analyzed in two possible orders (acd_1, acd_2 or acd_2, acd_1). In this way, there is a 50% probability that the ACS will behave wrongly. Furthermore, in such critical systems concerning with security and safety, the unpredictability of behavior and results is by no means desirable.

In the following sections, we introduce our proposed risk-adaptive ACS followed with the conflict analysis and resolution method for tackling the problems mentioned in this scenario.

5 Risk-Adaptive Access Control System (ACS)

In this section, the architecture of the ACS is presented. Its novelty lies in considering hierarchies of contexts and access control domains to manage different risks at different levels of criticality; while adopting break-glass policies in case of a crisis.

5.1 Risk-Adaptive ACS Architecture

The architecture of the risk-adaptive ACS is depicted in Fig. 1. While *MD* monitor $ATTR_{context}$, data streams are sent to the Safety Management System (SMS), which, in a MAPE loop [3]: 1) monitors the meaningful parameters and identifies the hazards $H_c \subseteq H$ if there are any; 2) analyzes $H_c \subseteq H$ and performs risk assessment to provide the description of the risk and its consequences; 3) plans the preventive strategies; 4) executes the strategies that can be automatically executed by the SMS (e.g., turn on alarms), and supports the execution of human-operated strategies (e.g., evacuation of an area).

Hazard $h_i \in H_c$ may lead to a set of risks $R_c \subseteq R$, each with the following attributes that constitute the *"risk description"*: *Type* as a unique name identifying the kind of the risk (e.g., fire); *Source* as the entity in the *EN* (device, machine, etc.) causing the risk; and *Location* refers to the $en_i \in EN$ affected by the risk. To simplify, we consider risks to be independent, as the dependency between risks can complicate the risk assessment procedure since dependent risks may have effects on one another.

Each $r_i \in R_c$ is connected to a set of consequences $C_{ci} \subseteq C$. In the analysis phase, the SMS also calculates the following attributes for each risk's consequence: *Type*, that is a unique name identifying the kind of the consequence, e.g., damage to the infrastructure, injury, death, etc; *Intensity*, namely the severity of the consequence, specified as a level; and *Probability*, namely the degree of uncertainty related to the occurrence of such consequence. In the planning phase performed by the SMS, the ACS receives the *risk description* and uses it to identify the *"Context"*. We consider a safety-centric approach to model the *Context*, considering different attributes of a risk and its consequences.

5.2 Hierarchical Contexts and Access Control Domains

Fig. 2 shows how *Contexts* are mapped to the *ACD* considering the hierarchy of *Contexts* and *ACD*. *Contexts* are defined in a hierarchy where the *Safe Context* defines the state of the environment that is considered as safe, namely with no risks. Contexts

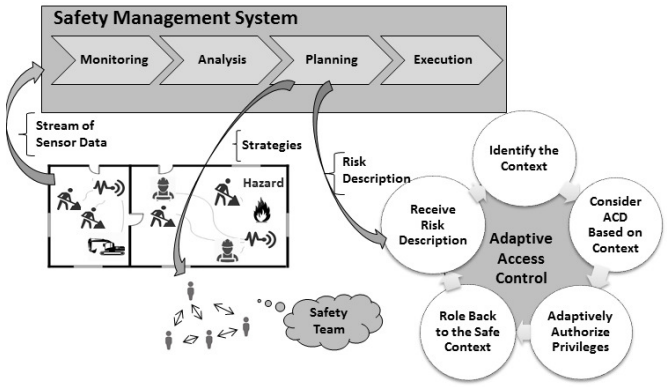

Fig. 1. Architecture of the risk-adaptive access control system.

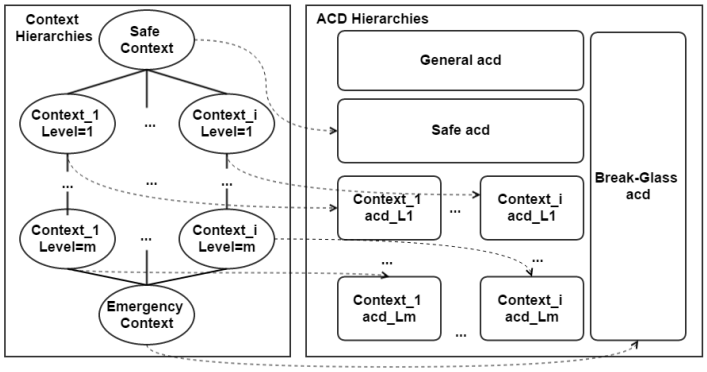

Fig. 2. Hierarchical levels of *ACD*.

$\{Context_1, ..., Context_i\}$, each refer to a risk-prone state of the environment, where different risks and consequences are present. Levels $\{Level_1, ..., Level_m\}$ are considered to represent the criticality of the *Context* that allows prioritizing its importance. For instance, a *Context* with $Level_4$ of criticality (e.g., fire) is considered with higher priority than a *Context* with $Level_1$ (e.g., dust in the air), hence deserving different priority of actions to face the potential risk. Finally, the *Emergency Context* is associated with the highest level of criticality and has the highest priority over all the *Contexts*.

To consider safety-centric contexts we consider two factors: First, based on the risk description $RD = \{r_i.Type, r_i.Probability, r_i.Source, r_i.Location, r_i.Consequence\}$, where $r_i \in R_c$, we define the *Context* which is a name assigned to it that represents the safety status of the environment. Then considering the $\{r_i.c_j.Type, r_i.c_j.Probability, r_i.c_j.Intensity\}$, where $c_j \in C_{ci}$, the *Level* is defined that illustrates the hierarchy of the contexts with respect to its criticality. Receiving the risk description from the SMS triggers the process of setting *Contexts* and calculating its *Level*.

Table 4. Notations for formal representations of ECA rules

Notation	Representation
ON	Operator catching an event
IF	Logical conditional operator for checking the conditions represented in the risk description
$\rightarrow$	logical then operator representing the action which is setting the proper contexts
$\wedge$	Logical AND operator
$\vee$	Logical OR operator
$\sim$	Logical NOT operator
$>$	Greater than
$<$	Less than
$==$	Equivalent to
$=$	Set to
$++$	Increment operator
$ADD(param, set)$	Operator for adding a value ($param$) to a set.
$REMOVE(param, set)$	Operator for removing a value ($param$) to a set.
$ISIN(param, set)$	A Boolean operator that checks if a value ($param$) exists in a set.

ECA Rules for Dynamically Selecting the Access Control Domains: To select the *Context* that best describes the safety status of the environment we consider a rule-based approach. Event-Condition-Action (ECA) rules have been commonly used for modeling context-aware behavior because of their flexibility and expressiveness. Moreover, adopting ECA rules enables defining contexts based on protocols and overall safety objectives of different organizations. We introduce formal notations for representation of ECA rules. Later on we introduce our XML implementation used for implementing and adopting them in applications. Table 4 lists the formal notations defined for representing ECA rules.

Calculating Context Level of Criticality: To calculate the level of criticality of the contexts, as mentioned before, we consider risk consequences and their probability, and intensity. Types of Consequences of risks can be categorized as: Loss of life; Injury; damage to infrastructure and physical resources; financial damage; and reputation damage. Moreover, the probability and intensity of the consequences are represented with qualitative measures, namely: Very high; high; medium; low; very low. The types of consequences are quantified by assigning a value to them which is usually based on the organization's protocols and views. For example, one organization may consider reputation damage more important than the financial damage and therefore assigns a higher value to it. And to quantify the probability and intensity we consider integer values starting with 1 to represent very low and 5 representing very high (more fine grained values can be used if needed). Table 5 represent the examples of quantitative values that can be considered for consequence type, probability and intensity. These values are defined based on their importance according the the organization views.

Table 5. Qualitative and quantitative scales for calculating $Context_{level}$

$c_j.Type$	$c_j.Probability$	$c_j.Intensity$	$Value$
Loss of life	Very high	Very high	5
Injury	High	high	4
Damage to infrastructure and resources	medium	medium	3
Reputation damage	Low	Low	2
Financial Damage	Very low	Very low	1

The *Context Level* is calculates as follows:

$$Context_{Level} = r_i.Probability \times \left(\sum_{j=1}^{n} \frac{c_j.Type \times c_j.Probability \times c_j.Intensity}{n} \right) \quad (1)$$

where $c_j \in C_{ci}$ and C_{ci} is the set of n detected consequences of $r_i \in R_c$.

To define the safe and emergency contexts, we define two thresholds based on the views and protocols of the organization, namely: T_{safe} which represents a threshold on $Context_{Level}$ below which is considered safe; and $T_{emergency}$ that a threshold on $Context_{Level}$ above which is considered an emergency situation. All other $Context_{Level}$ values in between T_{safe} and $T_{emergency}$ are normalized to an integer between $[1..m]$.

Mapping Contexts to ACDs: *Contexts* are mapped into different hierarchies of *ACD* that are sets including the security rules applied to control the access to objects under safety-related circumstances. As depicted in Fig. 2, we define a hierarchy of $acd_i \in ACD$. The *General acd* defines a subset of *SR* that is commonly applied regardless of the *Context*. This allows avoiding to repeat the common, shared security policies in different ACDs. The *Safe acd* includes a subset of *SR* that is applied in a *Safe Context*. Moreover, $\{acd_{L1}, ..., acd_{Lm}\}$ are defined for each $\{Context_1, ..., Context_i\}$ as *Context-specific acd* to include the subsets of *SR* that are applied in different *Contexts* at different *Levels*. When a *Context-specific acd* with L_x is applied, all the *Context-specific acd* with the lower levels for the same *Context* will also apply, while *SR* combination rules are in place that give the precedence to the *SR* at the higher level *Context-specific acd* when a conflict arises. Moreover, if a *Context* is considered as an *"emergency"* then not the *General acd* nor the other *Context-specific acd*s would apply and only the *"Break-Glass" acd* will be considered that includes a subset of *SR* that apply in an emergency situation.

Since we are mapping the ACDs to XACML Policies, mapping contexts to ACDs is facilitated. Here we make an example to clarify:

An Example: Considering a process industry where a risk description is received from SMS which indicates that there is a medium probability of risk of fire, with the risk source of gas pipes, in the warehouse. The consequences are estimated to be injury, with high probability and medium intensity, and damage to infrastructures and resources with high probability and very low intensity. According to the organization protocols, injury has an importance value of 4 while for damage to infrastructure and resources this value is 2. Also the thresholds are defined as: $T_{safe} = 24$ and $T_{emergency} = 110$. The $Context_{level}$ is calculated according to (1) and Table 5 which equals to 108. Since $Context_{level} > T_{safe}$ we are not in the safe context and similarly

as $Context_{level} < T_{emergency}$ we are not in emergency context either. If the calculation of $Context_{level}$ indicates a safe or an emergency context we set the context accordingly and skip the next step. Otherwise, which is also our case in this example, the value of $Context_{level}$ is normalized to an integer value between $[1..m]$. Here we consider $m = 5$ that leads to having five levels of criticality. Considering $T_{emergency}$ to get the highest possible level of criticality, $Context_{level}$ will normalize to 4.77 that is rounded to 5.

In the next step the *Context* should be specified using ECA rules:

$$
\begin{aligned}
&ON : r_i.Type == "Fire" \\
&IF : r_i.Location = "Warehouse" \wedge r_i.Source = "GasPipes" \\
&\rightarrow ADD("FWG", Context).
\end{aligned}
$$

The above mentioned ECA rule specifies that on the event that there is a risk of fire with the conditions that the location is the warehouse and the risk source is the gas pipes a context is set to FWG that is a unique name for this situation and its added to the set of active contexts.

6 Run-Time Conflict Analysis and Resolution

As shown previously, when there are several contexts active at the same time, conflicts may arise with the combination of SR in the corresponding ACDs. In the case of the Safe and Emergency contexts, we would not face this problem as in emergency context only the Break-Glass ACD is active, and in the safe context other risk-related contexts are not present and therefore we would not face unpredictable conflicts and the use of XACML combination rules suffices.

For conflict analysis and resolution we adopt XEngine [8] which fits pretty well to our needs, namely, it is efficient for a run-time approach. In fact, merging policies that have different XACML combining algorithms to produce appropriate results is quite a complex task, as we illustrated in an example in Section 4. The authors in [8] solve this problem by normalizing policies and converting all combining algorithms to "First-Applicable". The result of this procedure is a sequence of range rules $< predicate > \rightarrow < decision >$.

An issue is that XEngine merges policies with the same target, however, we need to merge policies (ACDs) with different targets that in our case are the contexts that are enabled at the same time. To resolve this issue, we propose creating a string with separators that contains the enabled Contexts and then, when a request arrives, all the SR in the corresponding ACDs are treated as they have the same target and therefore merged properly.

6.1 Request Analysis

Every request sent by a subject is analyzed by the *analyzeRequest* function, shown in listing 1.1. When this function is called, it checks if there are active Contexts; if one of the Contexts is "emergency", then the "break-glass" policy is selected. In general, a policySet set is generated and normalized, obtaining a set of range rules, from which we obtain a Policy Dependency Diagram (PDD) [8]. Starting from the PDD we generate a set of forwarding tables that let us compute the *effect* of the request.

```
analyzeRequest(Request req): String {
 if isContextEnabled(req.context))
  String[] activeContexts = getActiveContexts()
  String ACDActive
  if isEmergencyActive(activeContexts)
   ACDActive = getBreakGlassACD()
  else
   ACDActive = getActiveContextACD(activeContexts)
  RangeRule[] rangeRules = xacmlPolicyNormalization(ACDActive)
  PDD pdd = getPDD(rangeRules)
  ForwardingTable[] forwardingTables =
      constructForwardingTables(pdd)
  OriginBlock result = processRequest(req, forwardingTables)
  return result.decision
 else
  return "deny"
}
```

Listing 1.1. Request analysis main functions

6.2 Risk and Emergency Support Function

In Listing (1.2), the support functions called by *analyzeRequest* are shown: *isContextEnabled* checks if the context specified in the request is among the risk specific contexts that are active; *isEmergencyActive* returns true if one of the active contexts is "emergency"; *getActiveContexts* retrieves an array with the currently active contexts; *getActiveContextsPolicySet* enables obtaining a policySet integrating the ACDs that correspond to the active risk specific contexts; *getBreakGlassACD*, instead, returns the "break-glass" ACD, used in case of emergency.

```
isContextEnabled(String Context): Boolean {}
isEmergencyActive(String[] activeContexts): Boolean {}
getActiveContexts(): String[] {}
getActiveContextsPolicySet(String[] activeContexts): String {}
getBreakGlassACD(): String {}
```

Listing 1.2. Request analysis main functions

6.3 Normalization and Effect Processing

In Listing 1.3, functions corresponding to XEngine are reported: *xacmlPolicyNormalization* normalizes the policies in input, generating a set of *range rules*; *getPDD* builds a Policy Decision Diagram, on which a set of forwarding tables are created by the *constructForwardingTables*; finally, *processRequest* generates an *OriginBlock* (adopted to compute the *effect*), based on the request and computed forwarding tables.

```
xacmlPolicyNormalization(String policy): RangeRules[] {}
getPDD(RangeRule[] rangeRules): PDD {}
```

```
constructForwardingTables(PDD pdd): ForwardingTable[] {}
processRequest(Request request, ForwardingTable[]
    forwardingTables): OriginBlock {}
```

Listing 1.3. Request analysis main functions

7 Implementation

In this section we shortly describe the implementation details of the proposed risk-adaptive ACS and the conflict analysis algorithm. The ACS is developed employing open source technologies offered by WSO2. More precisely, Balana (an implementation of XACML 3.0) and WSO2 Identity Server, which implement XACML 3.0's data-flow model are used. To be able to use these frameworks for our purposes some modifications have taken place that is described in what follows.

7.1 Customization of the Policy Editor

WSO2 Identity Server is modified to adapt its PAP's (Policy Administration Point) user interface, to enable customization of XACML security rules, policies and policy sets based on our specific requirements. In the original WSO2 PAP Basic Policy Editor user interface, it was not possible to specify a target/condition on a different resource attribute than its identifier. Moreover, we needed to add contexts as the attribute of the Environment and to set it as the target of XACML Policy for defining context-specific ACDs. With the mentioned modifications we are able to automatically build XACML policy language to define ACDs based on any attribute of Subjects, Objects and the Environment.

7.2 ECA Rules Implementation

ECA rules are implemented via the following XML schema shown in Listing 1.4. We adopted similar notations to XACML policies for ECA rules for expressiveness, clarity and simplification of usage.

```
<metarule> :- <when> <if> <then>

<when> :- <when_anyof>
<when_anyof> :- <when_allof>+
<when_allof> :- <risk>+
<risk> :- name,<risk_parameters>
<risk_parameters> :- <risk_parameter>+

<if> :- <if_anyof>
<if_anyof> :- <if_allof>+
<if_allof> :- <condition>+
<condition> :- name,<condition_parameters>
<condition_parameters> :- <condition_parameter>+
<condition_parameter> :- value,type
```

```
<then> :- <actions>
<actions> :- <action>+
<action> :- <action_parameters>
<action_parameters> :- <action_parameter>+
<action_parameter> :- actionType, name, value, type
type :- "variable"|"immediate"
```

Listing 1.4. Meta-rule XML structure

7.3 Conflict Analysis and Managing Multiple Contexts

According to the proposed risk-adaptive ACS, when a new risk arrives and the system is in its "safe" state (i.e., safe context is active), then the string that identifies the context will be assigned to the environment context attribute ($en_i.Context$ where $en_i \in EN$). Otherwise, if the system is already in a risk state, the new risk-specific context will be appended to $en_i.Context$, according to the following notation:

$$en_i.context == "context_1\#level\&context_2\#level\&...\&context_n\#level"$$

Where *context* is the unique name of the context and *level* is the context's level of criticality. Thanks to this kind of notation, we can keep the environment's context attribute as a simple data type. When needed, we can read all the contexts that are activated, by simply splitting the string. This does not affect the efficiency much as this string does not get very long for each section of the environment assuming that no more that some limited number of risks happen at a time. At this point, the ACS (in its PDP component), will apply XEngine algorithm, combining all the policies that are triggered by the active contexts included in the $en_i.Context$ string. When a context is no longer active (which is detected by the SMS [3]), the context in the $en_i.Context$ string is removed. Then, the presented algorithms for conflict analysis and resolution will re-combine the currently active ACDs. If there are no more active risk-specific contexts left, the PDP considers general and safe ACDs.

8 Concluding Remarks and Future Works

In this paper we have presented our risk-adaptive ACS, based on the ABAC paradigm and XACML policy language. For each category of entity involved in the system (*subject*, *object* and *environment*), two types of attributes are considered: *security related* and *context specific* (the values of the latter depend upon the data received from the monitoring devices and is updated when there is a change). We have realized a hierarchical structure of safety-centric contexts, which lets us manage security rules specified for various risks of level of criticality using hierarchies of ACDs. To define the contexts ECA rules are considered that using the risk description and applying the predefined conditions identify the context.

Considering the dynamic combination of the ACDs may pose unpredictable conflict, at this point, we have adopted conflict analysis and resolution algorithm based on XEngine, to find and resolve possible conflicts among the ACDs' security rules that are applicable by the enabled contexts. To be able to response in a timely manner to emergency situations we have enabled the possibility to adopt Break-Glass security rules with minimum reasoning and no need for conflict analysis. Finally, when the

environment is in a safe state the ACS roles back to security rules that are normally applied.

One of the challenges regarding the security rules and the ECA rules adopted in this work is to automatically check their consistency. Moreover, to evaluate real-time efficiency of the proposed approach use-cases and scenarios should be considered and tested on the implemented ACS. As future works, authors will consider mentioned issues.

References

1. M. Al-Zewairi, J. Alqatawna, and J. Atoum. Risk adaptive hybrid rfid access control system. *Security and Communication Networks*, 8(18):3826–3835, 2015.
2. D. Fall, T. Okuda, Y. Kadobayashi, and S. Yamaguchi. Risk adaptive authorization mechanism (radam) for cloud computing. *Journal of Information Processing*, 24(2):371–380, 2016.
3. M. Fugini, M. Teimourikia, and G. Hadjichristofi. A web-based cooperative tool for risk management with adaptive security. *Future Generation Computer Systems*, 54:409–422, 2016.
4. S. Gusmeroli, S. Piccione, and D. Rotondi. A capability-based security approach to manage access control in the internet of things. *Mathematical and Computer Modelling*, 58(5):1189–1205, 2013.
5. C. G. Hoyos and B. Zimolong. *Occupational safety and accident prevention: behavioral strategies and methods*. Elsevier, 2014.
6. V. C. Hu, D. R. Kuhn, and D. F. Ferraiolo. Attribute-based access control. *Computer*, (2):85–88, 2015.
7. X. Jin, R. Krishnan, and R. S. Sandhu. A unified attribute-based access control model covering dac, mac and rbac. *DBSec*, 12:41–55, 2012.
8. A. X. Liu, F. Chen, J. Hwang, and T. Xie. Xengine: a fast and scalable xacml policy evaluation engine. In *ACM SIGMETRICS Performance Evaluation Review*, volume 36, pages 265–276. ACM, 2008.
9. M. A. Neri, M. Guarnieri, E. Magri, S. Mutti, and S. Paraboschi. Conflict detection in security policies using semantic web technology. In *Satellite Telecommunications (ESTEL), 2012 IEEE First AESS European Conference on*, pages 1–6. IEEE, 2012.
10. R. Roman, J. Zhou, and J. Lopez. On the features and challenges of security and privacy in distributed internet of things. *Computer Networks*, 57(10):2266–2279, 2013.
11. S. Schefer-Wenzl, H. Bukvova, and M. Strembeck. A review of delegation and break-glass models for flexible access control management. In *Business Information Systems Workshops*, pages 93–104. Springer, 2014.
12. I. Shamoon, Q. Rajpoot, and A. Shibli. Policy conflict management using xacml. In *Computing and Networking Technology (ICCNT), 2012 8th International Conference on*, pages 287–291. IEEE, 2012.
13. K. K. Venkatasubramanian, T. Mukherjee, and S. K. Gupta. Caac–an adaptive and proactive access control approach for emergencies in smart infrastructures. *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, 8(4):20, 2014.
14. D. Yan, J. Huang, Y. Tian, Y. Zhao, and F. Yang. Policy conflict detection in composite web services with rbac. In *Web Services (ICWS), 2014 IEEE International Conference on*, pages 534–541. IEEE, 2014.