RESEARCH PAPER | Engineering | Volume : 5 | Issue : 8 | August 2015 | ISSN - 2249-555X

Processing Flows of Information at Emergency Situation

KEYWORDS: floods, earthquakes, hurricanes, and man-made disasters

C. KAMAL
M TECH (cse), DEPARTMENT OF CSE
MLR Institute of Technology, Dundigal, Hyderabad.

N. ARAVIND KUMAR
Assistant professor, Department of CSE, MLR Institute of Technology, DUNDIGAL, Hyderabad.

ABSTRACT *During natural disasters or emergency situations, an essential requirement for an effective emergency management is the information sharing. In this paper, we present an access control model to enforce controlled information sharing in emergencysituations. An in-depth analysis of the model is discussed throughout the paper, and administration policies are introduced to enhance the model flexibility during emergencies. Moreover, a prototype implementation and experiments results are provided showing the efficiency and scalability of the system.*

1 INTRODUCTION

In the last years, natural catastrophic events, e.g., floods, earthquakes, hurricanes, and man-made disasters, e.g., airplane crashes, terrorist attacks, nuclear accidents, highlight the need for a more efficient emergency management. In particular, attacks of September 5, 2001, have shown that the lack of effective information sharing resulted in the failure to intercept the terrorist attacks [1]. This example points out the need of a more efficient, timely and flexible information sharing during emergency management. Indeed, during an emergency there is often the need to access resources that are not allowed during the normal system operations. However, such downgrading of object security classification should be controlled and temporary. To cope with these requirements in [5], we propose an Access control model to enforce controlled information sharing in emergency situations. Our model is able to Enforce flexible information sharing within a single organization through the specification and enforcement of Emergency policies. Emergency policies allow the instantiation of temporary access control policies (tacps) that override regular policies during emergency situations. More precisely, each emergency is associated with one or more tacp templates, describing the new access rights to be enforced during specific emergency situations. In general, in emergency management scenarios the response plans are defined by experts on the field based on regulations and laws and based on reports resulting by the emergency preparedness phase, during which emergency managers conduct a risk assessment .We believe that all these documents represent a solid base from which emergencies, emergency policies, and emergency obligation scan be specified.

In this paper, we propose an extended version of the model in [5]. One of the main extensions concerns administration policies. Indeed, this is a crucial task in every application scenarios, but in case of emergency management is even more strategic due to the need of areal-time adjustment of the authorization state upon the modification of security requirements. Moreover, these specification of emergency policy requires both a security expert as well as an expert of the domain of the consider emergencies. This is captured in our model through the definition of proper scopes that limit the right to stream emergency policies only to specific emergencies.

The paper also presents an in-depth analysis of the checks performed by our system to ensure policy correctness, which where only roughly sketched in [5]. Finally, the prototype Implementation presented in [5] has been extended to Implement the correctness validity checks and administration policy enforcement. We also report new performance tests on the prototype, more extensive, and detailed than those presented in [5].The remainder of the paper is organized as follows: Section 2 presents an overview of the model [5]. Policy correctness is analyzed in Section 3. Section 4 presents administration policies. The prototype implementation and performance evaluation is provided in Section 5. Section 6surveys related work, whereas Section 7 concludes the paper.

2 EMERGENCY INFORMATION SHARING

To enforce flexible information sharing during emergencies, normally authorized. Moreover, it is often the case that specific actions should be performed to manage the Emergency. To fulfill both these requirements, the mode presented in [5] supports tacps to be activated during emergencies and obligations that have to be fulfilled when an emergency is detected. The connection of an emergency with the corresponding tacps and obligations is modeled by emergency policies. A language, called Core Event Specification Language (CESL), is used to define events describing the beginning/ending of an emergency. The formal syntax of CESL operators is reported in [5].

Definition 2.1 (Emergency description):
An emergency emg is a tuple (init, end, time-out, identifier), where init and endure emergency events specified in CESL, such that init denotes the event triggering the emergency and end is the optional event that turns off the emergency, time-out is the time within the emergency expires even though end has not occurred. Identifier is an attribute belonging to both the schemes of the event type corresponding to init and end events. Note that the identifier plays a key role in that it ensures the connection between in it and end events (see [5] for further details) as shown in the following example; which also illustrates the reference scenario used throughout the paper. This has been chosen to show how our model works in a challenging domain, where the number of emergencies and related emergency policies is large and the level of policy granularity is high. Even if we are aware that this is not a typical domain for emergency management (e.g., disaster management), we decide to select it because it gives us the opportunity to provide more complex examples of emer-

Volume : 5 | Issue : 8 | August 2015 | ISSN - 2249-555X

gency descriptions and policies.

Example 2.1 (Reference scenario).
Patients are hospitalized at home, in a special clinic or in a hospital. Each of these structures provides patient treatments through specialized equipment able to ensure a real-time monitoring of patient vital signs. Data gathered by the monitoring equipment are collected by the CEP to automatically detect emergency situations. More precisely, we suppose that, every 30 seconds, each sensor sends the systolic pressure of patients to the monitoring system in the Vital Signs stream of tuples (pressure,…, patient id). Hypertension emergency can be defined as follows:

Hypertension Emergency {
init: VS1 v1;
VS1 ¼ _(pressure > 140)(Vital Signs);
end: VS2 v2;
VS2 ¼ _(pressure _ 120)(Vital Signs);
timeout: 1;
identifier: patient_id;}
The emergency starts when the diastolic pressure of a Patient is higher than 140 mmHg, and it ends when the Pressure of the same patient (i.e., with the same patient_id) returns to less than or equal to 120 mmHg. When the Hypertension Emergency is detected for patient 1,the following emergency instance is created.
HypertensionEmergencyInstance1 {
emg: Hypertension Emergency;
identifier: 1;}

The HypertensionEmergencyInstance1 is deleted when Hypertension Emergency ends for patient 1. Our model enforces controlled information sharing during emergencies through tacps. More precisely, because different instances of the same emergency might require different tacps, we associate with an emergency a tacp template, that will be properly instantiated when an emergency is detected.

3 .EMERGENCY POLICY CORRECTNESS

The main function of emergency policies is the enforcement of the corresponding tacps/obligations upon emergency detection. More precisely, emergency policy enforcement consists of two main steps: 1) the creation/deletion of the corresponding emergency instances and 2) the consequent creation/deletion of instances of the corresponding tacps. As discussed in Section 5, emergency activation/deactivation is

a time-consuming operation. Therefore, a particular attention has to be paid in properly defining the init and end emergency events to ensure that, even if syntactically well-defined, they will not imply a simultaneous activation and deactivation of an emergency. In general, this type of error occurs when the two sets of tuples satisfying in it and end

Events are not disjoint. Indeed, in this case, the arrival of just one tuple may cause the simultaneous creation and deletion of the corresponding emergency and tacp instances. Let us consider, as an example, an emergency specification, where nit: temp _ 37 and end: temp _ 39. In this case, the arrival of tuple t such that t:temp ¼ 38 results in the simultaneous

Creation and deletion of the corresponding emergency

And tacp instances.2 We formally define this problem by showing also the correctness and enforcement in the following sections.

Here and in the following, we use dot-notation to indicate fields of events, emergencies or policies.

4 EMERGENCY POLICY ADMINISTRATIONS

Emergency management is a complex task, that we believe requires distributing the rights of create/modify emergency policies among different subjects, called emergency managers. Indeed, people in charge of the planning of response activities for emergency situations have a strong expertise in issues dealing with the particular field originating the emergency. For example, in the hospital scenario the head of cardiology ward has the best profile to indicate, which Activities have to be performed for cardiology emergencies, but not to determine the response plan for a breathing emergency. we make use of the concepts of emergency scope and tacp scope, which are formally defined as follows:

Definition 4.1 (Emergency scope):
An emergency scope is at tuple (event, streams, operators), where event2 finit; end; both, streams is a set of stream names, and operators is a setoff CESL operators. Given an emergency description e and an emergency scope emg_scope, we say that e is valid w.r.t. emg_scope, if the init (end or both, respectively) event is defined on a subset of the streams specified in emg_scope.streams, by using a subset of CESL operators specified in emg_scope. operators. Definition 4.2 (Tacp scope). A tacp scope is a tuple (sbj, obj, priv, ctx, and obl) where: sbj, obj, and ctx are subject, object, and context specification, respectively; priv and obl are a set of allowed privileges and actions, respectively.

Given a tacp template tacp and a tacp scope a cp_ scope, we say that tacp is valid w.r.t. a tacp_scope if: the subject(object, respectively) specification of tacp identifies a subset of subjects (objects, respectively) identified by tacp_scope.sbj(tacp_scope.obj, respectively), the set of values for a context attribute identified by a context specification of a tacp is a subset of the values identified by tacp_scope.ctx and theprivileges (obligations) in tacp. priv (tacp.obl) is a subset of those privileges (obligations, respectively) identified in tacp_scope. Priv(tacp_scope.obl). Based on emergency and tacp scopes, we can now formalize the emergency administration policies, as follows:

4.2 Administration Policy Enforcement

The enforcement of emergency administration policies is carried out each time a user defines or modifies an emergency policy, with the aim of verifying whether the new policy satisfies at least an administration policy. In case an emergency policy is not valid w.r.t. the specified administration policies, a set of rewriting strategies are applied, aiming to redefine the invalid emergency policy so as to make it valid w.r.t. at least one of the specified administration policies. Every time an emergency policy is rewritten, awarding is sent to the emergency manager who has defined the policy to inform him about the rewriting operation and, in

case of bad rewriting, to manually correct the policy. In case an emergency policy is not valid w.r.t. any administration policy and rewriting is not possible, the emergency policy is discarded and the policy issuer is warned. When a user defines/modifies an emergency policy, the validity of the new emergency policy is verified by Algorithm 2.

Volume : 5 | Issue : 8 | August 2015 | ISSN - 2249-555X

Algorithm 2.ValidateEmergencyPolicies
input: ep, the new emergency policy to be validated
input: u, the user which is trying to define ep
output: ep, ; or a list of valid rewritten emergency policies
1 Let EAPR be the Emergency Administration Policy Base;
2 rwEPs¼ ;;
3 for each eap 2 EAPR do
4 <r, np> ¼ Check EmergencyPolicy (u, ep, eap);
5 if r ¼ Valid then returns ep;
6 if r ¼ ValidAfterRw then
7 rwEPs ¼ rwEPs[{np};
8 Warn (u, ep, eap);
9 end
10 if rwEPs¼ ; then Warn (u, ep);
5 return rwEPs;

Algorithm 2 takes as input an emergency policy ep and the user u who is trying to define it. Algorithm 2 check Sep against each administration policy eap in the Emergency Administration Policy Repository EAPR (Lines 3-9) by using the Check EmergencyPolicy function(line 4). This function takes as input u, ep, eap and returns a pair <r, np> with one of the following values:<Valid, ep>, if ep is valid w.r.t. eap, <Invalid, ; > , if ep is not valid w.r.t. eap, <ValidAfterRw, np>, where np is are written emergency policy, if ep is not valid but there writing strategy can be applied. If r = Valid, then Algorithm 2 returns ep (line 5). If r = ValidAfterRw, then the rewritten emergency policy np is stored into the rwEPs set (line 7) and user u is informed that the emergency policy he/she has defined has been rewritten (line 8). In case ep is not valid, when Algorithm 2 has analyzed all the emergency administration policies, it returns rwEPs, which could be empty or contain the setoff rewritten emergency policies (line 5). In case rwEPs is empty, the ep emergency policy is not inserted into the policy base and the user u is warned about the wrong definition of ep (line 10).

Function CheckEmergencyPolicy (u, ep, eap)
1 Let np = (tacp, emg, obl) be initialized empty;
2 EmgChk = ChkEmgScope (ep:emg, eap:emg scope);
3 <r, np.tacp> = RwTacp (ep:tacp, eap:tacp scope);
4 if u 2 eap: adminsbj ^ ep: obl _ eap: obl ^
EmgChk ¼ true ^ r ¼ Valid then
5 return <Valid, ep>;
6 if u 62 eap:adminsbj _ ep: obl \ eap: obl ¼ ; _
EmgChk ¼ false _ r ¼ Invalid then
7 return <Invalid, ;> ;
8 np.emg ¼ ep:emg;
9 np.obl ¼ ep: obl \ eap: obl;
10 return <ValidAfterRw, np>;

Function Check EmergencyPolicy. These function first checks if the emergency description ep:emg is valid w.r.t. the emergency scope eap:emg scope (line 2) through function ChkEmgScope. Then, it calls unction Rw Tacp (line 3), which takes as arguments the tacp template contained into theinput emergency policy and the tacp scope of the input Administrative policy and returns a pair <r; np:tacp> that can have one of the following values: <Valid; ep:tacp>, if ep.tacp is valid w.r.t. eap.tacp_scope, <Invalid; ;>, if ep.tacp is not valid w.r.t. eap.tacp_scope, <V alidAfterRw; np:tacp>,

Where np.tacp is a rewritten tacp, if ep.tacp is not valid w.r.t. eap.tacp_scope, but it can be rewritten into the valid policy np.tacp. Then, CheckEmergencyPolicy verifies whether the user is among the authorized users in eap, obligations specified in ep are a subset of those authorized in eap and both the tacp template and the emergency description contained into the input emergency policy are valid w.r.t. The corresponding scope (line 4). If all these conditions are satisfied, then CheckEmergencyPolicy returns <Valid; ep>(line 5), otherwise it checks if there is at least a condition to

consider ep not rewritable into a valid policy, that is, if u is not among the authorized users in eap, or obligations required in ep are disjoint from obligations allowed in eap,or Check Emergency Scope returns false or Rw Tacp returns Invalid (line 6). If at least a condition holds, then CheckEmergencyPolicy returns <Invalid; ;> (line 7).

Function ChkEmgScope.
Function Chk Emg Scope takes as input an emergency emg and an emergency scope and returns true or false whether emg is valid or not w.r.t. scope. Depending on scope.event content, the function checks if the streams over which init/end or both of them is defined are a subset of the streams contained in the scope and if the operators used in init/end or both of them are a subset of the scope operators. In case these checks succeed, the Function returns true, otherwise it return false.5Function RwTacp (t,s)

1 Let n ¼ (sbj, obj, ctx, obl) be initialized empty;
2 rw ¼ false;
3 <res, np.sbj> ¼ RwTacpSbj (t.sbj,s.sbj);
4 if res ¼ Invalid then return <Invalid, ;>;
5 if res ¼ ValidAfterRw then rw = true;
6 <res, np.obj> ¼ RwTacpObj (t.obj,s.obj);
7 if res ¼ Invalid then return <Invalid, ;>;
8 if res ¼ ValidAfterRw then rw ¼ true;
9 if t.priv \ s.priv= ; then return <Invalid, ;>;
10 if t.priv 6_ s.priv then
5 rw ¼ true;
12 np.priv ¼ t.priv \ s.priv;
13 end
14 <res, np.ctx> ¼ RwTacpExp (t.ctx,s.ctx);
15 if res ¼ Invalid then return <Invalid, ;>;
16 if res ¼ ValidAfterRw then rw = true;
17 if t.obl \ s.obl= ; then return <Invalid, ;>;
18 if t.obl 6_ s.obl then
19 rw ¼ true;
20 np.obl ¼ t.obl \ s.obl;
21 end
22 if rw ¼ false then return <Valid, t>;
23 else return <ValidAfterRw, np>;

5 .PROTOTYPE IMPLEMENTATION AND TESTS:

The prototype is implemented in Java on top of a Stream Base CEP platform [28]. We describe how the prototype works during the three most important phases:1) specification of emergency descriptions, tacp templates and emergency policies; 2) emergency activation/deactivation; and 3) user access.

5.1 Performance Evaluation

In this section, the performance results of the prototype system are discussed. The experiments were run on an Intel Core i7 2.00-GHz CPU machine with 4-GB RAM, running Windows 7. The prototype implements the architecture explained in Fig. 1; therefore, we carried out tests on every step of the emergency life cycle. In this section, we report results on overall time for emergency activation/deactivation and user access time. We refer to Appendix D, available in the online supplementary material, for tests

Volume : 5 | Issue : 8 | August 2015 | ISSN - 2249-555X

on each single step (i.e., event detection time, emergency creation time, tacp creation time, emergency deletion time, tacp deletion time) and for comparison between PP detection time and activation/ deactivation time. Before presenting the experimental results, we provide details on the data set.

5.1.1 Data Set

To carry out the experiments on emergency detection, Activation, and deactivation, we developed an emergency events generator. By means of this generator, we can create a specific number of init and end events by varying their complexity, which is measured in terms of number of operators (i.e., selection, aggregation and join operators) contained into the event.

As shown in Fig. 2, in case of complexity 1, the generated event takes as input a unique stream, over which it evaluates one selection and two aggregations. From this unique input stream, it generates both in it and end events. With a complexity of two, the event contains two input streams, two selections, four aggregations, and two join operators. In general, in case of complexity n, the number of input-streams is n, the number of selections is n, the number of aggregations is 2n, and the number of join operators is P nffiffiP i¼1 2i (see, as an example the case of complexity4 in Fig. 2).

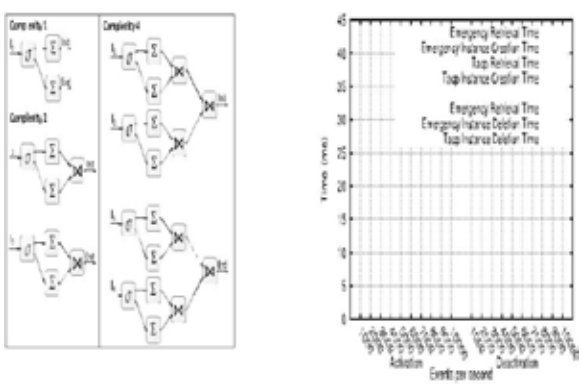

Fig. 1.System architecture.

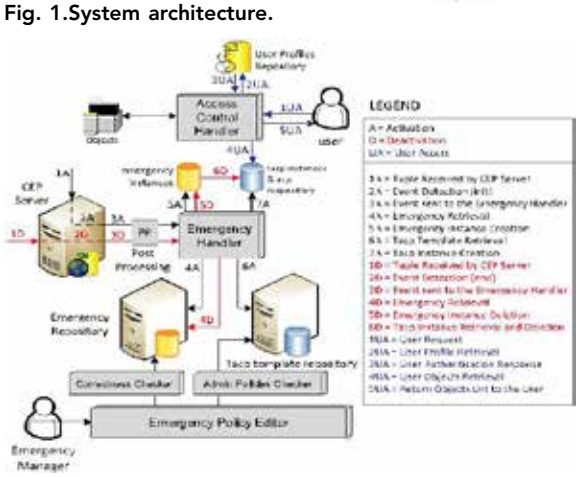

Fig. 2.Emergency event complexity.

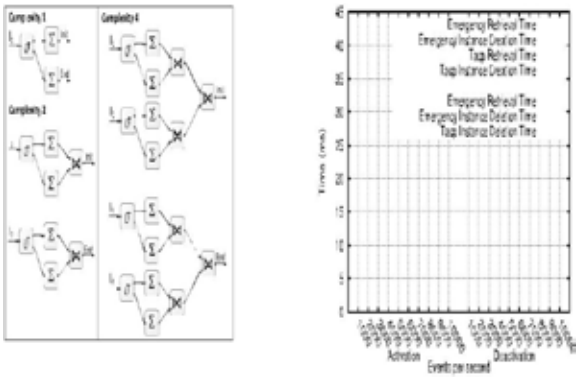

Fig. 3activatiom/deactivation time

5.1.2 Activation and Deactivation Overall Time

The overall activation time represents the time elapsed Between the detection of an emergency and the effective activation of the corresponding emergency policy. This is given by

1. The time needed to retrieve the emergency related to the triggered init event (emergency retrieval time),
2. The time for the creation of the corresponding

Emergency instance (emergency instance creation time),3. the time necessary to retrieve the tacp template Related to the emergency (tacp template retrieval Time), and4. the time to create the corresponding tacp instance(tacp instance creation time).The overall deactivation time represents the time elapsed between the detection of a tuple satisfying an end event and the effective deactivation of the corresponding emergency policy.

Similar to activation.time. Emergency activations and deactivations. During experiments, the tuples rate varies from 1.000 to 10.000 tuples per Second, which means that the number of activated Emergencies per hour varies from 3.600.000 to 36 million? Considering, for instance, that the daily volume of 95 calls for New York city is 30.000 [7], we believe that our experimental numbers are large enough to guarantee high performance in a real emergency management system.

6 RELATED WORKS

Our model enforces fine-grained access control with attribute-level granularity. Many models have been pro-posed in the literature, in support of fine-grained access control, for instance models derived from ABAC or the XACML standard . A remarkable model supporting fine-grained access control in a healthcare domain is C-TMAC presented. This approach allows team-based access control by also integrating contextual in formation. In [5], we intentionally gave a high-level definition of the paper, we adopt RBAC-A. We believe the above-mentioned models can be adopted in our system instead of RBAC-A. However, none of them support emergency detection through CEP technology, which is a total novelty in access control systems.

7 .CONCLUSIONS:

In this paper, we proposed an extension of the emergency access control model presented in [5] with the possibility of defining administration policies, i.e., which subjects are enabled to define emergency policies and over which scope. Moreover, we have implemented an extended version of the prototype presented in [5], and we have car-

Volume : 5 | Issue : 8 | August 2015 | ISSN - 2249-555X

ried out an extensive set of test to check what is the impact of emergency policies into an access control system. A set of correctness checks have also been defined to avoid useless activation/deactivation of emergencies.

REFERENCE [1] "The 9/5 Commission Report," technical report, Nat'l Commission on Terrorist Attacks upon the United States, July 2004. | | [2] J.G. Alfaro, "N.: Management of Exceptions on Access Control Policies," Proc. 22nd IFIP TC-5 Int'l Information Security Conf. (IFIPsec '07), pp. 97-108, 2007. | | [3] C. Ardagna, S. De Capitani di Vimercati, S.Foresti, T. Grandsons, S. Jajodia, and P. Samarati, "Access Control for Smarter Healthcare Using Policy Spaces," Computers and Security, vol. 29, pp. 848-858, 2010. | | [4] M.Y. Becker, "A Formal Security Policy for an NHS Electronic Health Record Servce," Technical Report UCAM-CL-TR-628, | | [5] B. Carminati, E. Ferrari, and M. Guglielmi, "Secure Information Sharing on Support of Emergency Management," Proc. IEEE Third Int'l Conf. Privacy, Security, Risk and Trust (PASSAT), and IEEE ThirdInt'l Conf. Social Computing (Social Com), pp. 988-995, Oct. 2011.