A Fine-grained Access Control Technique to Protect Sensitive Healthcare Data in Emergency Cases with Dynamic Security Policies

Luong Khiem Tran[1], Ha Xuan Son[1,2], Tran Khanh Dang[1], and Kim Tuyen Le[1]

[1] Ho Chi Minh City University of Technology, Ho Chi Minh city, Vietnam,
khiem111189@gmail.com, hxson@ctuet.edu.vn, khanh@hcmut.edu.vn,
tuyenltk@hcmut.edu.vn
[2] CanTho University of Technology, Can Tho city, Vietnam

Abstract. One primary requirement for the healthcare industry is prioritizing the provision of information to support the patient's health. For this reason, access control mechanisms for health systems to regulate and limit disclosure of data are often ignored in the event of an emergency. This phenomenon, called "Break The Glass" is a popular model for healthcare systems, but from a security point of view it represents a serious weakness. In fact, malicious users can take advantage of this principle to gain unauthorized access and privileges. There are a number of solutions offered in previous research, but these approaches still contain some limitations. These restrictions have not been a complete replacement for the BTG principle yet in an emergency situation. In addition, it is so difficult that previous approaches can be applied to the actual situation as the main barrier comes from the complexity and the flexibility of the healthcare system. This paper propose Model of Access Control for HealthCare system (ACDP-HC Model), aiming to better regulate exceptions while ensuring data availability. Our solution is based on allowing for a flexible mix of policies created by admin and patient. Besides, the mechanism supporting dynamic policy also adds the flexibility to the ACDP-HC. In the evaluation, we compared ACDP-HC Model with previous approaches. As a result, ACDP-HC better supports than that. We also have developed a proof-of-concept for this mechanism and we show some experimental results based on difference scenarios.

Keywords: access control, healthcare system, break the glass, dynamic policy, rewriting request, policy evaluation

1 Introduction

As sensor networks, distributed system, and Internet of Things (IoTs) are the physical endpoints of a health care application, a number of traditional healthcare strategies has been becoming online healthcare systems service. In fact, global healthcare organizations make a change of their data into Electronic Patient Health Records (EPHR). In Barack Obama's speech at the American Medical Associatihis 2009, he mentioned that all the medical information should be

stored securely in a private medical record [23]. Accordingly, a patient's information can be tracked from multiple doctors, even that patient has changed his/her hospital and specialists. However, the interaction among patients, doctors, insurance companies, and pharmaceutical ones supported by healthcare systems are challenging problems. In fact, the sensitive nature of the information managed by health care requires the balance between two contrasting needs: demands for data to ensure the highest quality to the patients "availability" and for sensitive data security (chronic diseases, mental health issues, psychiatric care, sexual behavior, fertility issues, abortion status, and HIV status). Therefore, the core concern of the management systems is to prevent sensitive data from unauthorized access. According to that, access control (AC), a primary type of mechanism, is being applied by healthcare systems to protect the data. However, the traditional access control models are rigid, so it is challenging to apply them to the medical environment. Nevertheless, one of the most specific unbroken requests of medical data is "nothing interferes with the delivery of care" [10], with restricted access control would likely be ignored in emergency situations or other similar exceptions. For instance, in urgent ones, a nurse can request (and must be authorized) access to the data, which "normal" conditions she cannot view. This phenomenon is known as "break the glass" (BTG).

Numerous studies have been done for applied BTG, in terms of emergency situation, as a compulsory requirement in healthcare systems [11]. The model "Break The Glass" is kept separated from the basic access control model suggested by Petritch et al. [16]. The model allows users to override the consequence of access control; thus, a user can access non-authorized resources. According to Ferreira et al. [5] the third option of BTG, in addition to grand and deny, grants permission to authorized users to break the glass instead of denying access. Additionally, Ferreira introduced a different BTG policies application in comparison with traditional ones, offering users unlimited access and responsibility under unsecured situations [6]. In addition, some obvious flaws were found in this model. In the BTG state, users can not only override access control results (the user is granted greater authority), but they also ignore the system constraints, resulting in significant damage to the system.

Variety approaches to prevent BTG's weaknesses, in the event of emergency or exception, have been suggested such as: minimizing possible cases lead to BTG, protecting medical data by cryptography functions, and defining the special policy for the emergency situation (BTG policy, policy spaces). Nonetheless, the above mentioned methods have not completely overcome BTG in emergency situations. In this paper, the main aim is an introduction to an approach to prevent current shortages by providing privacy policy in both Normal and Emergency Situations, followed two criteria: the availability of data and the secured nature of the sensitive data. Our approach is to provide policy in both Normal and Emergency Situations. In order to achieve this aim, two groups of users are considered in parallel existence: administrator participating in the system management to provide `system policies` and patients contributing to create `sensitive data policy`. By this approach, administrators and patients

are allowed to modify (insert, delete and edit) their policies in the real-time.The primary goal is to increase the flexibility of the system and to adapt quickly to the complex healthcare institution where people change their roles and departments frequently [6]. A flexible access control model is applied to support special cases, which is followed by the approach in previous researches [19, 20]. The contributions of this work includes the following:

- Minimize emergency situations leading to BTG.
- Introduce appropriate rules and policies for medical data.
- Provide access control mechanisms in an appropriate dynamic policy environment for healthcare systems.
- Provide a timely response mechanism when a policy update occurs.
- Implementing our proposed theoretical model.

The rest of the paper is organized as follows: In the next section, we provide relevant background material. In section 3, we demonstrate our proposed model and implementation. In the following sections, we evaluate our implementation based on security analysis, compared with the other models and measuring performance. The next section is devoted to the prior work and then will conclude the paper with a summary of our results and discuss opportunities for further research in conclusion.

2 Background

2.1 Policy Definition

This section is focused on an overview of the policies included in the system. Policies can be formatted in JSON [15] or XML [17] depending on the defined system. With the initial policies structure, an update is difficult to be detected by system. A transformation of policies is presented to flattens the hierarchical structure of a policy while preserving the semantics of the original policy.

Policies are defined in a schema $\langle Att, Dom \rangle$ with Att is the set of `Attributes` and $Doms$ are assigned to the `Attribute Domains`. An attribute assignment denotes an expression of policy in extensional form and maps each attribute to a (possibly empty) set of values in its domain. Each policy will have certain attributes that need to be protected. For example, patients' name are prohibited to be seen by other parties, except - doctors and nurses in charge of the patients' medical care. In order to monitor the updating of policies in the system, a proper policy management mechanism, where each individual policy value (security attributes, constraint values) is managed, is necessary. In traditional approach, all policies in the system are stored in a fixed place, and are called when necessary. By this approach, to manage policies in a dynamic policy environment is challenging.

Definition 1: A single rule is converted into `Attribute Type` (at) and `Applicable Constraint` (ac). The at and ac are linked through logical operations. A policy consists of many rules and is linked together through the

Combining Algorithm Rule and so does a policy set, excepting the way of linking, through the Combining Algorithm Policy.

Each policy partitioned the data into two parts: the data area protected by the policy and the data area unprotected by that policy. The non-policy area will be divided into two sub-sections: access requests for evaluating the constraints produces an error and access requests for the target (condition) evaluates to no match. Consequently, when the system receives an access request, it results in three specific values: Applicable, Not Applicable, and Indeterminate corresponded to appropriate policy, inappropriate one and satisfy missed evaluation, respectively.

Definition 2: The Applicability Space of the policy is composed of 3 elements: Applicable Space, Not Applicable Space and Indeterminate Space. As defined above, a Applicability Space include all possible access requests. Applicability Space is represented as a triple $\langle AS, NAS, IS\rangle$, in which AS, NAS contains the access requests for which the target and condition evaluates to `Match` (True) and `NoMatch` (False) respectively, and IS contains the requests for which evaluates the constraints producing an error.

The value of the Applicable Space depends on the value saved in the Effect element (`Permit / Deny`). Consequently, the Applicable Space is subdivided into two cases, Applicable Space Permit and Applicable Space Deny. Assessing access requirements for a policy in the system leads to an access decision. The decision space of a policy divides the policy space into access classes that are evaluated according to the same decision.

Definition 3: The Decision Space (DS) of a policy element is a tuple $\langle DS_P, DS_D, DS_{IN}, DS_{NA}\rangle$ so that each member of this is a set of access requests evaluated for `Permit, Deny, Indeterminate,` and `Not Applicable`, respectively. The indeterminate decision space DS_{IN} is a triple $\langle DS_{IN_P}, DS_{IN_D}, DS_{IN_{PD}}\rangle$ represents unidentified decisions for all three cases IN_P, IN_D, and IN_{PD} using `Combining Algorithm`: `Override Permit, Override Deny`.

In the definition of decision space, the indeterminate decision space is described as a triple of two special `Combining Algorithms`. In other words, the value of the indeterminate decision space is the set of values for all three components. To be able to map to the Decision Space as a set of four values, the DS_{IN} is replaced by three subsets: $\langle DS_P, DS_D, DS_{IN_P} \cup DS_{IN_D} \cup DS_{IN_{PD}}, DS_{NA}\rangle$. Hereafter, the usage of notation DS_{IN} or notation $\langle DS_{IN_P}, DS_{IN_D}, DS_{IN_{PD}}\rangle$ depending on the context of use. The following section presents how a policy defined in XML/JSON is converted to a combination of *at* and *ac*.

2.2 Encoding Policy

Policy Example : In "**normal situation**", a doctor or nurse only have access to the patient information they receive on arrival only if they are present in the patient's treatment department need access. The other user group is insurance companies with limited access such as access to hospital's fee and ID number of patients; or the pharmacy with limited access such as the ID number and prescription drugs of patients. In "**emergency situation**",when and only if

the doctor or nurse who takes care of a patient is absent and the others want to access the medical data of this patient, it must be satisfied: either by the patient's agreement or possibly a family member/whom the patient fully trusts[1]

In this example, the policy set includes policy elements and possibly combines these policies using `first-applicable (fa)`. One way to model the policy elements is to represent (the negation of) these constraints as `Deny` rules. In addition, a default policy is introduced with effect `Permit` and combined using `deny-overrides (dov)`. According to `dov`, the policy yields decision `Permit` only if none of the `Deny` rules is applicable. This policy set is represented in the notation as follows:

$$P(set) : \langle p(set)[com\_al\_policy\_ID], p_1, p_2, \ldots, p_n \rangle \tag{1}$$

Where $p[com\_al\_policy\_ID]$ is the ID of combining algorithm policy and $p_1, p_2, \ldots, r_n$ denoted from `policy 1` to `policy n`. As in the case of the *policy set structure*, the *policy structure* is defined as follows:

$$P : \langle p[com\_al\_rule\_ID], r_1, r_2, \ldots, r_m \rangle \tag{2}$$

The information of patients are included, for example, ID, name, gender, age, address, phone, drugs, health history, department, relation, hospital's fee, AvailableDoctorNurse, patientEmerAgree and relatPatientEmerAgree. AvailableDoctorNurse determines whether the presence of doctors/nurses are present at the hospital. In the meantime, patientEmerAgree and relatPatientEmerAgree determine the patient's agreement and family member agreement in emergency situation. The original policy set is represented in policies and rules as follows:

$p(set)[fa] : resource : "patients' \ record" \wedge action : "read"$
$p_1[dov] : subject : "nurse" \wedge "doctor"$
$\quad r_1[Deny] : AvailableNurseDoctor = \textbf{false}$
$\quad r_2[Deny] : patient \notin List\_Caring\_Patients$
$\quad r_3[Permit] : \textbf{true}$
$p_2[dov] : subject : "insurance \ companies"$
$\quad r_4[Deny] : patients' \ information \notin \{ID, \ hospital's \ fee\}$
$\quad r_5[Permit] : \textbf{true}$
$p_3[dov] : subject : "pharmacies"$
$\quad r_6[Deny] : patients' \ information \notin \{ID, \ drug\}$
$\quad r_7[Permit] : \textbf{true}$
$p_4[dov] : subject : "nurse" = "doctor"$
$\quad r_8[Deny] : AvailableNurseDoctor = \textbf{true}$
$\quad r_9[Deny] : patientEmerAgree = \textbf{false} \wedge relatPatientEmerAgree = \textbf{false}$

[1] **Note:** The normal situation is used for all patient data areas and is managed by admin, while the emergency situation is managed by the patient, depending on the type of data that the patient needs different defenses that the policy defines. These policies are called if the data zone it is accessed in an emergency situation. Without this policy, the patient is unnecessary to request personal data protection. Data is easily accessed by responsible doctors or nurses.

$r_{10}[Permit]$: **true**

As introduced in **definition 1** below, we represent the attribute type at_i and applicability constraints ac_j defined from the policy set and policy element:

at_0: *"patients' record"* ∈ **resource**; at_1: *"read"* ∈ **action**;

$at_2, \ldots, at_5$: {*doctor, nurse, insurance companies, pharmacies*} ∈ **subject**;

at_6, at_7: $\forall v \notin \{patients' information, patients\}$;

ac_0: $\forall v \in$ **patients' information.AvailableNurseDoctor**, $v =$ **false**;

ac_1: $\forall v \in$ **patients**, $v \notin List\_Caring\_Patients$

ac_2: $\forall v \in$ **patients' information**, $v \notin \{ID,\ hospital's\ fee\}$;

ac_3: $\forall v \in$ **patients' information**, $v \notin \{ID,\ drug\}$;

ac_4: $\forall v \in$ **patients' information.AvailableNurseDoctor**, $v =$ **true**;

ac_5: $\forall v \in$ **patients' information.patientEmerAgree**, $v =$ **false**;

ac_6: $\forall v \in$ **patients' information.relatPatientEmerAgree**, $v =$ **false**;

The decision spaces for the whole policy set example is constructed from above *ats* and *acs* same as the process in the paper [19] with the approach proposed in [21]. The encoding result is used for analyzing the policy set with different changes in the dynamic policy context. The example is used through this paper, especially in the experiments Section 5.2.

Obligations Os contain one or more obligation(s) O. An obligation is an action that takes place after a decision has been reached to either Permit or Deny. It is mapped to ACDP-HC within the context of policy and policy set according to the following syntax:

$Os ::= Obligation\_set$

$O ::= \langle OID, FF, AID, DT, V \rangle$

Where *Obligation_set* is the set of obligation (O). OID is the ID identifying of obligation, FF is the $fulfill$ (`On/Off`) attribute is a boolean function that is used as a key of determine when the obligation must be enforced and must be either permit or deny. AID is the attribute ID of the obligation to be carried out. DT is the data type and V is the data value. If the policy or policy set being evaluated matches the FF (`On`) attribute of its obligations, then the obligations are passed to be enforced otherwise obligations are ignored.

Target T is an objective and is mapped to ACDP-HC within the context of rule, policy and policy set according to the following syntax:

$T ::= \langle S, A, E, Rs \rangle$

Where S is the set of subjects, A is the set of actions, E is the set of environments, and Rs is the set of resources.

A rule R is the most elementary element of a policy. A rule contains rule conditions, target and rule effect. It is mapped to ACDP-HC according to the following syntax:

$R ::= \langle ID, RC, T, RE \rangle$

Where ID is the identify of rule, RC is the set of rule conditions, T is the set of targets, and Re is the rule effect.

A policy P is a single access control policy. It is expressed through a set of rules. A policy contains a set of rules, rule combining algorithm, target and obligations. It is mapped to ACDP-HC according to the following syntax:

$P ::= \langle ID, SR, RCA, T, Os \rangle$

Where ID is the identify of policy, SR is the set of rules, RCA is the rule combining algorithm, T is the set of targets, and Os is the set of obligations.

Policy set PS is created by the set of policies P. PS may contain other policy sets, policies or both. It is mapped to ACDP-HC according to the following syntax:

$PS ::= \langle ID, SP, RCP, T, Os \rangle$

Where ID is the identify of policy set, SP is the set of rule, PCA is the policy combining algorithm, T is the set of targets, and Os is the set of obligations.

3 Approach

3.1 Types of Policy

The policy in the system consist of two types: system policy (managed by administrator) and sensitive data policy (managed by the patient).

System policy is aim at raising system security level and avoiding illegal access from unauthorized users. Noticeably in this policy, when the users fully satis es the system policy constraints, the user can access the requested data while ignoring the sensitive data policy. If the users meet none of the system policy requirements, the system converts to the sensitive data policy evaluation.

Sensitive data policy is entitled to create data protection policies by patients. In fact, it depends varied data protection needs of different patient groups. For example, if a patient is a celebrity the protected information includes name, address, phone number, email. Other patient prefers that used medication method and medical record are protected from the insurer because of the fear of insurance rejection due to infectious diseases. In these example cases, sensitive data policy is absolutely defined as the policy in the emergency situation. The sensitive policy is inapplicable if information protection is unnecessary.

3.2 Dynamic Policy

The Dynamic Policy section discusses how Update Management (UM) works, policy updates is categorized by different update type with separate actions, including: insert policy, delete policy and edit policy.

Insert policy: A new policies are implemented through the Policy Administrator Points (PAP). Either admin or patients could add new policies if they need to secure certain data areas that have not protected yet in the former policies. Actually, this new addition does not provide an influence over the results returned to the users because the policies in the system must satisfy the requirements of two features: non-conflict and non-redundancy.

Delete policy: When a policy is removed from the system, UM will define whether the policy grants access to any access requests. If the result is negative, delete policy does not produce investigation outcome. In this case, the UM does not send the request to the PDP or PEP. Conversely, UM sends a request to the PDP to locate another policy in order to replace the deleted policy.

Edit policy: The editing is adjusted by a policy owner. UM will define whether the policy grants access to any access requests. If the result is positive, the following action leads to three smaller instances: insert rule, delete rule, and edit rule. Otherwise, it ignored. The details of these three cases are described as follows:

- **Insert rule:** both the admin and the patients express their hope to achieve additional constraints or conditions to the policies. Hence, UM sends a request to PEP with the aim of rewriting requests. Afterwards, PEP takes them into consideration. If there is a lack of attributes, PEP forward the authentication request as Obligation Service (OS). The users (access requesters) must provide the attribute/evidence (for a certain period of time) before rewriting process. PEP rewrites the query by adding the newly identified evidence to the original request and moves it to the PDP to re-evaluate the process with the updated policy request.
- **Delete rule:** the users including (admin and patients) want to restrict constraints or conditions over the policies. PDP re-evaluate the query with updated policy.
- **Edit rule:** This case occurs when admins or patients want to change the constraints or conditions. Similar to delete rule, PDP re-evaluates the query with updated policy.

4 Implementation

4.1 ACDP-HC Model

This section initiates the proposed Model of Access Control for HealthCare system (ACDP-HC Model). Different users of a healthcare system are classified into 4 groups: Admin, Patient, Internal User and External User.

Admin: the administrator of the system through the system policies. The system policies explain the conditions for accessing medical data in the "Normal Situation".

Patient: a person who is granted permissions to create policy for sensitive data if they find it necessary. Patient can restrict access to other users such as insurance companies and pharmacies, or even the doctors and nurses who are not responsible for patient treatment in the "Emergency Situation".

Internal User: a person who plays a role as a doctor or nurse. They are the two essential groups of the healthcare system. In "Normal Situation", internal users are only allowed to view information of their patients. In "Emergency Situation", it is imperative that they fulfill all of the patient's requirements as defined in sensitive data policy before accessing sensitive data.

External User insurance company and pharmacy. In "Normal Situation" insurance company and pharmacy interact with medical data through the system policy. In "Emergency Situation" insurance company and pharmacy can not access patient information, unless the patients grant them access.

The ACDP-HC Model consists of six components:

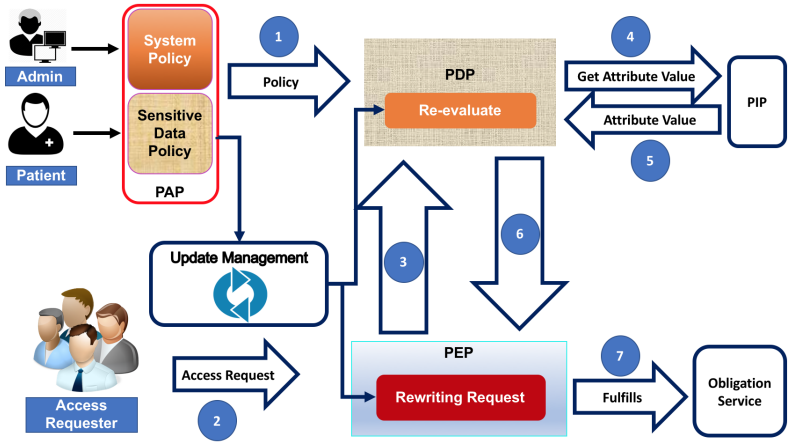

Fig. 1. ACDP-HC Model

Policy Administrator Points (PAP): The system entity stores all policies (system policy and sensitive data policies). Additionally, PAP converts the entire policy to be stored as *at* and *ac*. PAP's work is to provide new policies for PDP and policy updates for UM.

Policy Decision Points (PDP): The system entity evaluates the process, which is the most significant one of the system, where finds the most suitable policy and assigns permissions to access request. In addition, PDP containing a sub-entity named re-evaluate request. Its duty is to rewrite access request when UM calls for.

Policy Information Points (PIPs): The system entities stores attribute values. PIP provides the required attribute values during the evaluation process.

Policy Enforcement Points (PEPs): The system entity receives user requests and formats the request as the tuple $\langle$`attribute - value`$\rangle$. Then forward to the PDP in oder to evaluate the process. In addition, PEP containing a sub-entity is named rewriting request. Its duty is rewriting access request when UM calls for. If to rewrite the query because of insuficient attributes or evidence is impossible, PEP sends an authentication request to OS. The OS send these requests to demand the users provide missing attributes and evidence before rewriting the access request.

Update Management (UM): The system entity always listens to changes in the PAP. These updates are categorized by UM and then send rewriting or re-evaluating requirements to PEP or PDP.

Obligation Service (OS): The system entity sends a request to the access requester is demanded to fulfill all such requests for a certain period of time before sending the entire return to PEP.

Figure 1 describes the ACDP-HC model in 7 prominent steps:

1. The admin or patient sends their policies to the system. These policies are converted into concatenation via the logical operations between *ac* and *at*.

2. Internal or External User sends a request to access the system.
3. PEP sends all requests after being reformatted for the PDP. Here, PDP evaluate the process.
4. PDP sends a request for attribute value for PIP.
5. PIP responds to the value attributes that PDP looks for.
6. PDP returns the outcome to PEP after the evaluation process ends.
7. After receiving the response from PDP, PEP will fulfill Obligation Services (if any) before executing the users request.

During the seven steps, ACDP-HC model shows the steps for monitoring the policy update process on the PAP. If any changes occur to the available policies in the system, UM proceeds to classify those changes and determine next step by re-evaluating or rewriting the request.

4.2 Algorithm

Algorithm 1 Optimized Dynamic Policy Evaluation

```
1: Require: PAP: pap instance; Q: request instance
2: ⟨decision, non_applied_policies, applied_policy⟩ = evaluate(Q, PAP)
3: changes = getPolicyChanges(PAP, applied_policy)
4: switch changes.type do
5:     case DeletePolicy
6:         return evaluate(Q, PAP, exclude : non_applied_policies)
7:     case InsertPolilcy
8:         return decision
9:     case EditPolilcy
10:        if editConstraints then
11:            return evaluate(Q, PAP, only : applied_policy*)
12:        else if deleteConstraints then
13:            return evaluate(Q, PAP, only : applied_policy*)
14:        else if insertConstraints then
15:            tempAtt = getLackAtt(Q, applied_policy*)
16:            v = Obligation(tempAtt)
17:            Q* = rewiteRequest(Q,v)
18:            return evaluate(Q*, PAP, only : applied_policy*)
19:        end if
```

The Algorithm 1 classifies a policy change from their property (*Delete, Insert* or *Edit policy*). The **Require** of this method includes the PAP instance and the access request (line 1). After the policy evaluation process, the result is stored as the tuple $\langle decision, non\_applied\_policies, applied\_policy \rangle$ where *decision* returns either `Permit` or `Deny`, *non_applied_policies* contains the evaluated policy and applied_policy is the policy assigned to the access request (line 2). In line 3, *changes* is a parameter storing type of change. If the temp's value is "*Delete*",

the method which finds a new policy with the goal of replacing the original policy is deleted. In this process, the *non_applied_policies* policies are ignored (line 5 and line 6). In the type of "*Insert*" the new policies are stored in the PAP and the system ignores *Insert* type (line 7,8). The case of "*Edit*" is divided into three smaller instances: insert rule, delete rule, and edit rule. Both delete and edit rule are re-evaluated by the access request with edited policy "*applied_policy**" (from line 10 to line 13). For the insert rule, the attributes or evidence are found if it does not exist in the request. If not exist, PEP will send an authentication request to OS. Otherwise, *tempAtt* returns null (line 16). Consequently, the Obligation sends these requests to force the users to provide missing attributes and evidence before rewriting and evaluating the access request (line 17 and 18). See [19] for more detail.

5 Evaluation

5.1 Comparison:

In this section, based on several criterias, we compared the ACDP-HC model with the STEM-RBAC model [8], the extension of the conventional RBAC access control policy - team based access control (TMAC) [9] and, implementations of the Attribute-Based Access Control (ABAC) - XACML v3.0 [17], The characteristics of these models are shown in Table 2.

Table 1. Characteristics comparison

	STEM-RBAC	**TMAC**	**XACML v3.0**	**ACDP-HC**
Fine-Grained Controls	Medium	Medium	High	High
Robustness	Low	Low	High	High
Dynamic Policy	Static	Static	Static	Dynamic
Practicality	High	High	Low	High
Context Information	Medium	Low	High	High

Fine-Grained Controls: It represents a special characteristic with flexible access control, which supports defining user permissions for the system after successful authentication. It is desirable to realize the ease of management without additional or complex manipulation of the access permissions. Both ACDP-HC models and XACML v3.0 can support high fine-grained access mechanisms because both models allow the definition of policies to the level of Attribute.

Robustness: It depends on parameters like flexibility.In the proposed approach, this requirement is maximally supported by dynamic policy mechanism. In the other models, XACML v3.0 is more flexible because, in addition to supporting spatial and temporal attributes, it supports other attributes.

Dynamic Policy: This feature is shown by whether the owner policy is allowed to change the policy in real time or not. According to Dunlop et al. [4],

a system is supposed to support the dynamic policy only if the following criteria are fulfilled:

- Policy includes a variable which contains a reference to run-time or location.
- Policy is able to be altered during request time.
- Ability to create, update or delete policy during the request time.
- Ability to dismiss the policy during the request time.

Considering these criteria, only the ACDP-HC model supports dynamic policy.

Practicality: It denotes how easy to enforce an access control system in a real-time system. The STEM-RBAC, TMAC models and ACDP-HC model are designed specifically for the healthcare system. Currently, there are many implementations of XACML v3.0. However, in order to integrate this model into the healthcare system, further research is necessary. Hence, practicality of XACML v3.0 is lower than the others.

Context Information: This characteristic shows how much the access control model can handle the user context. In other words, this characteristic defines whether permission rights are affected by changes in contexts. The TMAC model supports merely team components. Other contexts such as object condition is excluded. On the other hand, ACDP-HC model supports situation components, including composed of user context and object context. For example, when changes occur in the user's context , the user's attributes can be updated dynamically.

5.2 Experiments

This section describes the experiments are conducted to simulate the policy changes in the healthcare context and profile the process of evaluating requests in the context. The XACML policies, requests input and the developed simulation will be illustrated before the running profiling is analyzed and discussed.

The policies and requests are constructed from attributes in three categories Subject, Resource, Environment and Action which are presented in Figure 2. The object-oriented approach is used to construct Subject and Resource attributes. Those are *User.Id, User.Role* and others for Subject. For Resource, those are *Patient.Record.AssignedDoctor, Patient.ManagementInfo.Id* and other such attributes. From the approach in [14], policies are constructed based on two kinds of constraint. One kind of constraint is between an attribute and a value. Another kind of constraint, relationship constraint is between two attributes in different categories. Connection lines in Figure 2 are instances of this kind of constraint. The Axiomatic Abbreviated Language for Authorization (ALFA)[2] is used to specify policies to reduce the verbosity of XML format then guarantee the correctness of the generated policies in the XACML 3.0 standard. Figure 4 illustrates the policy set having the User Emergency Policy created by a patient is in higher priority in request evaluation than the Emergency Policy created by the System Administrator. Figure 5 illustrates changes to the User Emergency

[2] https://www.axiomatics.com/blog/tag/alfa/

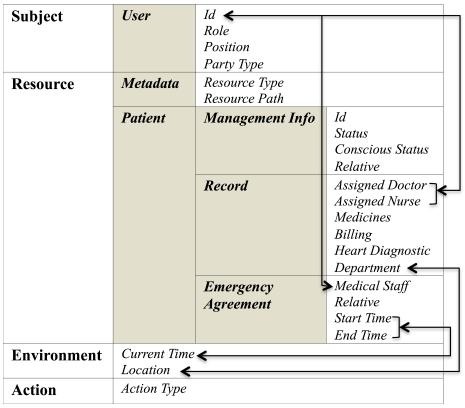

Fig. 2. Healthcare Attributes

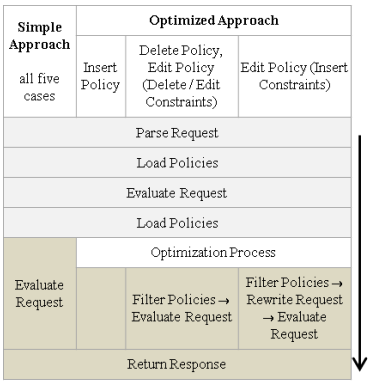

Fig. 3. Profiled Actions

Policy in five cases: the policy Deleted by the Admin, another policy Inserted by the patient and the policy Edited (Delete constraints, Edit constraints and Insert constraints) by the patient. The policies and requests are loaded and run successfully in both open source XACML implementations Balana and OpenAZ[3].

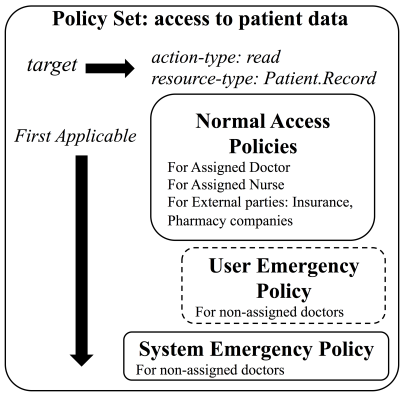

Fig. 4. PolicySet for Patient Data

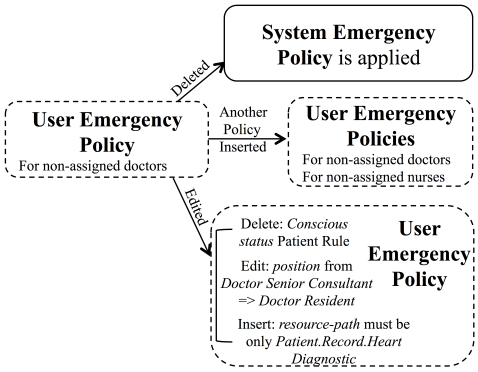

Fig. 5. Changes to the User Emergency Policy

The dynamic policy environment is simulated in Java[4] wrapping the Balana XACML framework as the static policy evaluation engine. In the dynamic policy environment, before returning the response, it is required to check if there is any policy change event. If there is no change, the response is proceeded to the client.

[3] https://github.com/khiem111189/balana
https://github.com/khiem111189/incubator-openaz
[4] https://github.com/khiem111189/research_group

If there are changes, there are two approaches to apply. The **simple approach** is re-evaluate the request with new policy changes. The **optimized approach** is illustrated in Algorithm 1. There could be a loop of checking policy change event and re-evaluate the request. Five cases in Figure 5 are used for experimenting those two approaches. In these experiments the policy change event happens only once.

The running experiments are profiled into process actions illustrated in Figure 3. The **optimized approach** introducing the *Optimization Process* action brings advantages on relinquishing the re-evaluation for the Insert Policy case; reducing the complexity of policies for the re-evaluation in the Delete Policy and Edit Policy case or improving the throughput by re-writing the original request with addition attribute values. However, the action *Filter Policy* does not reduce the total expending time of *Evaluate Request* actions less than the total one for the **simple approach**. The reason is that Balana evaluates request mostly one millisecond with different experimented policy sets. The system configuration for the experiments is a 64-bit machine with 8GB of RAM and 2.8 GHz Intel Core i5 CPU running macOS High Sierra. Moreover, actions inside the *Optimization Process* may cause new potential security problems to the whole evaluation process in comparison to only re-evaluating the original request of the **simple approach**.

6 Related Work

In traditional access control models such as Discretionary Access Control (DAC), Mandatory Access Control (MAC) or Role-Based Access Control (RBAC), the user privileges must be defined in advance and set a limited number of changes throughout the system implementation. Therefore, a great number of studies have extended the RBAC model in order to meet specific requirements in the medical setting. Garcia-Morchon and Wehrle have suggested a Context-Aware Role-Based Access Control (CA-RBAC) model [7] in order to raise awareness of context and to adapt its security properties in order to ensure the users' safety. The disadvantages of CA-RBAC model is the lack of preventive or detection mechanisms as well as a verification process to check the correctness of user's access in emergency context. Similarly, Georgakakis et al. introduced Spatio temporal emergency role based access control model (STEM-RBAC) [8]. The emergency access mechanism was recommended by the author. Additionally, STEM-RBAC model allows exception-albeit controlled access in the event of an emergency, using role hierarchies. A modular security monitor is switched on to monitor abnormal access (in case of an emergency). Kabbani et al. [12] introduced a Dynamic Authorization model based on the XACML v3.0 model. The Dynamic Authorization model adds to the flexibility situations of traditional XACML modeling and addresses the changes that occur in the healthcare system. In this model, the situation changing detection function is always enabled to receive change information from the external environment and to select the appropriate situation. The weakness of Dynamic Authorization model, how-

ever, is that all situations must be defined in advance and it does not support dynamic policy.

Yu et al. [22] proposed a Fine-Grained Distributed Data Access Control (FDAC) model based on Attribute-Based Encryption. A network controller, containing access structures, acts as a central distribution center and distributes keys to the users in FDAC. This approach may reduce the possibility that BTG leads to emergencies or exceptions. However, FDAC model is based on a centralized approach because only the network controller can perform the key management. If the network controller is compromised, provision in the network is not secured, which is a single point of failure. To avoid a single point of failure, Ruj et al. [18] proposed a multi-authority Attribute-Based Encryption (MABE) - a based access control plan. Their goal is to provide complete distributed data access control using a number of Distribution Centers (DCs). However, this approach does not explain in detail how to combine all access structures. Without a combined approach, the users must store all access structures to access different types of data from the sensor network. Maw et al. [13] proposed an Adaptive Access Control (A2C) access control model. The A2C model obtains priority and behavioral monitoring to provide effective access control for medical data, In A2C model ,users can override denied access when unexpected events occur. In addition, the trust model of user behavior is used to test the user's actions, location, and time, except the user's behavior. Without a trusting behavior model, access decisions can not be conducted effectively (limited). Some access control models for medical data such as FDAC, A2C and MABE use cryptographic methods to store data and control access to data. Nevertheless, systems need to predefine access, roles, as well as policies prior to deployment. Prioritizing possible needs for accessing real-world applications is challenging in these models. Obviously, unanticipated situations might occur spontaneously. For practical applications, the system needs to be flexible enough to make decisions on data access based on unusual circumstances beyond normal circumstances.

Ardagna et al. [2] introduced a new approach addressing the problem of balancing strict nature of traditional access control systems. The author defines the policy space regulating access to medical data and describes how policies are identified and enforced in each space and their combinations. The author approach governs all access; otherwise, they will fall into a "BTG policy", and aim to better handle "abnormal" access requests [1]. In this paper, the description defines the data space of a policy into four different spatial regions: Access (P+), Access is denied (P-), Planned Exceptions (EP), and Unplanned Exceptions (EU). Regretfully, this approach only supports the combining algorithm as "First-Applicable" for policies. This limitation influences the design of scenarios between policies because to anticipate all the activities (normal and urgent situations) that may occur in the healthcare system is impossible. Another disadvantage is that unauthorized users can access an emergency situation in some sensitive instances [3]. This model needs some secondary policies for each case to meet the privacy requirements.

7 Conclusion

This paper has proposed risk reduction strategies when emergency situation occurs. The approach based on authorized access is granted by the patients through creating sensitive data policies. If an unauthorized access to medical data is detected, all the operations from the users will be recorded and be analyzed. The strength of the proposed system is that it allows admins and patients freely to change policies during evaluation process or to grant access to another user. Besides, the healthcare system is complicated because each user can change positions, roles, and attributes on a regular basis. Supporting flexibility to a maximum extent is the main goal in our proposed model. Furthermore, in comparition to prior models, ACDP-HC model provides solution for current shortages. The measurements during the implementation indicate that ACDP-HC model is a promising approach for further application.

The forthcoming study will focus on implementing a mechanism of detecting conflict and redundancy when updating policies or adding new ones. Moreover, policy templates will be created to support admin better and provide additional features. The main aim is to identify changes to the external environment in order to provide hints for admin and patients to change policy. In addition, applying the system in a specific hospital in the future is considered as well.

References

1. Ardagna, C.A., et al.: Access control for smarter healthcare using policy spaces. Computers & Security 29(8), 848–858 (2010)
2. Ardagna, C.A., et al.: Regulating exceptions in healthcare using policy spaces. In: IFIP Annual Conference on Data and Applications Security and Privacy. pp. 254–267. Springer (2008)
3. Balasubramaniam, J., Fong, P.W.: A white-box policy analysis and its efficient implementation. In: Proceedings of the 18th ACM symposium on Access control models and technologies. pp. 149–160. ACM (2013)
4. Dunlop, N., Indulska, J., Raymond, K.: Dynamic policy model for large evolving enterprises. In: Enterprise Distributed Object Computing Conference. pp. 193–197. IEEE (2001)
5. Ferreira, A., et al.: How to securely break into rbac: the btg-rbac model. In: Computer Security Applications Conference. pp. 23–31. IEEE (2009)
6. Ferreira, A., et al.: How to break access control in a controlled manner. In: Computer-Based Medical Systems. pp. 847–854. IEEE (2006)
7. Garcia-Morchon, O., Wehrle, K.: Modular context-aware access control for medical sensor networks. In: Proceedings of the 15th ACM symposium on Access control models and technologies. pp. 129–138. ACM (2010)
8. Georgakakis, E., et al.: Spatio temporal emergency role based access control (stem-rbac): A time and location aware role based access control model with a break the glass mechanism. In: Computers and Communications. pp. 764–770. IEEE (2011)
9. Georgiadis, C.K., Mavridis, I., Pangalos, G., Thomas, R.K.: Flexible team-based access control using contexts. In: Proceedings of the sixth ACM symposium on Access control models and technologies. pp. 21–27. ACM (2001)

10. Grandison, T., Davis, J.: The impact of industry constraints on model-driven data disclosure controls. In: Proc. of the 1st International Workshop on Model-Based Trustworthy Health Information Systems (MOTHIS) (2007)
11. Joint, N.: Cocir/jira security and privacy committee (spc). Break-glass: An approach to granting emergency access to healthcare systems (2004)
12. Kabbani, B., et al.: Specification and enforcement of dynamic authorization policies oriented by situations. In: New Technologies, Mobility and Security (NTMS). pp. 1–6. IEEE (2014)
13. Maw, H.A., Xiao, H., Christianson, B.: An adaptive access control model for medical data in wireless sensor networks. In: e-Health Networking, Applications & Services (Healthcom), 15th International Conference on. pp. 303–309. IEEE (2013)
14. Medvet, E., Bartoli, A., Carminati, B., Ferrari, E.: Evolutionary inference of attribute-based access control policies. In: International Conference on Evolutionary Multi-Criterion Optimization. pp. 351–365. Springer (2015)
15. Nguyet, T.Q.T., et al.: Using json to specify privacy preserving-enabled attribute-based access control policies. In: International Conference on Security, Privacy and Anonymity in Computation, Communication and Storage. pp. 561–570. Springer (2017)
16. Petritsch, H.: A generic break-glass model. In: Break-Glass, pp. 37–50. Springer (2014)
17. Rissanen, E., et al.: extensible access control markup language (xacml) version 3.0. OASIS standard 22 (2013)
18. Ruj, S., Nayak, A., Stojmenovic, I.: Distributed fine-grained access control in wireless sensor networks. In: Parallel & Distributed Processing Symposium (IPDPS). pp. 352–362. IEEE (2011)
19. Son, H.X., Dang, T.K., Massacci, F.: Rew-smt: A new approach for rewriting xacml request with dynamic big data security policies. In: International Conference on Security, Privacy and Anonymity in Computation, Communication and Storage. pp. 501–515. Springer (2017)
20. Son, H.X., Tran, L.K., Dang, T.K., Pham, Y.N.: Rew-xac: an approach to rewriting request for elastic abac enforcement with dynamic policies. In: Advanced Computing and Applications (ACOMP). pp. 25–31. IEEE (2016)
21. Turkmen, F., den Hartog, J., Ranise, S., Zannone, N.: Formal analysis of xacml policies using smt. Computers & Security 66, 185–203 (2017)
22. Yu, S., Ren, K., Lou, W.: Fdac: Toward fine-grained distributed data access control in wireless sensor networks. IEEE Transactions on Parallel and Distributed Systems 22(4), 673–686 (2011)
23. Zhang, R., Liu, L., Xue, R.: Role-based and time-bound access and management of ehr data. Security and Communication Networks 7(6), 994–1015 (2014)