Chapter

An Intelligent Access Control Model

Shadha Mohamed Sulaiyam ALAmri

Abstract

Cybersecurity is a critical issue as the world is moving towered IR4 era (Industrial Revaluation 4.0) where technology is involved, and access to the internet is an imperative need. The traditional computing systems are not able to meet the huge computing demand and growing data (Big-Data). Therefore; new technologies have been evolved such as cloud computing. This chapter is exploring the need for a dynamic access control approach to enhance the Cybersecurity. The scope in this chapter is focusing on IaaS (Infrastructure as a Service) layer of cloud computing. The research approach aims to enhance the basic ABAC (Attribute-Based Access Control) model by adding a context-aware feature and SoD principle. The enhanced model called **ABACsh**. This proposed enhancement is implemented through a framework based on AI (Artificial Intelligent) to meet the requirements of dynamic systems. The framework is tested in the OpenStack testbed. The results show better performance in the term of computation speed.

Keywords: Cybersecurity, AI, ABAC, formal logic, IaaS, cloud computing, OpenStack

1. Introduction

Industrial revolution 4 (IR 4.0) utilizes technology in different aspects. As per the world economic forum, three principle technology drivers in the industrial production: connectivity, intelligence and flexible automation where big data is one of the value drivers and IoT is one of the scale-ups enablers [1]. To summarize one scenario of the embedded technology in the industry is the implementation of IoT systems. Internet of Things (IoT) starts with a collection of sensors used to collect information from the surrounded environment. For example, a temperature sensor used to collect the atmosphere temperature during the day by taking three reads for six months for the purpose of studying climate change. The collected data will be sent to central storage such as cloud computing technology to get the advantage of accessing the data anywhere and anytime. There is a need for a network connective that allows distributed components to be connected. Mostly the collected data is a type of big-date as they might collect temperature reading from a different site in the globe and for a long time might be years. That big-data requires some analysis where the traditional analytical system might not manage to absorb its huge records, therefore; there is a need to utilize the features of artificial intelligence filed in data-science. This example shows how several technologies are used in order to be used in analyzing the big-data collected from different sites.

As there are distributed systems and the internet connection is used, cybersecurity becomes a critical aspect, especially when there are some economic benefits. There are many security principles which might be tackled in order to enhance the cybersecurity of the systems, however, access control is one of the major aspects as there will be a need to restrict the access to the system as there is a distributed environment when it comes to IoT deployment. One of the optimal access control models to be used in this case is attribute-based access control model (ABAC) [1, 2].

This chapter introduces intelligent attribute-based access control model tested in the cloud computing environment. Section 2 discusses the introduced enhancements to the basic ABAC model. Section 3 illustrates how inelegant is introduced in the proposed ABAC. Finally, an empirical experiment is demonstrated in Section 4 where OpenStack (cloud environment) is used to discuss the efficiency of the proposed approach.

2. The proposed enhancement in ABAC model

2.1 Context-aware analytical study

Context-aware system has verity of definitions based on the study scope. In access control field, context-aware allowing a dynamic permission to access an object based on some attributes related to the user context [3]. The context can be extracted from the system environment by using 5W1H (who, where, what, why, when and how) [4]. The context attributes are a finite set which reflect the system and differs from the attributes related to the subject and the object as per the researcher in [5]. However, other authors consider the rule enforcement through Attribute-Based Access Control (ABAC) is based on the attributes of both the subject and the object [6]. Therefore; this section is investigating context-aware concept in access control.

An ubiquitous application with RBAC extension has been investigated by Kim's [3] where state checking matrix is used to build a context-aware agent. Two cases are defined to deploy context-awareness. The first one is through giving privilege up-on the user context, such as location and time. The second one is changing resource permissions up-on the system information, such as network bandwidth and memory usage. Another work proposed by Kim in this filed, called CIAAC (Context Information-based Application Access Control) [4]. CIAAC designed to separate processing logic and business from context awareness and access control policy. CIAAC add flexibility to business application which support dynamic access control policy. This feature allows to satisfy the demand of external security environment. However, the potential drawbacks of CIAAC have not yet been evaluated. Another technique was proposed by Li in his thesis [7] to meet the scope of mobile cloud environment based on Attribute-Based Encryption (ABE). Li defines context-aware terminology to cover the user context-information in addition to the environment such as location and time.

As per the literature, encryption techniques such as ABE introduce several limitations which effect the overall system efficiency such as the overhead caused by bilinear pairing due to its heavy computation [8]. In addition to that ABE cannot attain fine-grained control [9]. Another related work done by AL Kukhun [10] considering pervasive systems where XACML language is used to build a model to extend RBAC that can facilitate context-aware features. However, RBAC extinctions approaches do not satisfy usability, situation awareness, and improving access opportunities. It can be observed that location and time are used as context-aware parameters in most related work on context-aware access control models. Liu and

Wang [11] present the Fine-grained Context-aware Access (FCAC) model for Health Care and Life Sciences (HCLS) using specific communication technology based on linked data. FCAC is based on two main components: an ontology base, and access policy with XACML.

It is observed from the state of art that context attributes are linked to the system environment rather than subject-attributes or object-attributes. Venkatasubramanian et al. [12] investigate context-aware to distinguish between the traditional authorization models and their proposed criticality-aware as they take into consideration the context of the whole system. Their criticality-aware (CAAC) is based on RBAC concepts. Choi [13] used access-aware in cloud computing. Choi recommends an ontology-based Access Control Model (onto-ACM). Compared to C-RBAC (Context-aware RBAC), onto-ACM can grant the role inheritance by administrator and user, whereas C-RBAC grants the role by administrator only.

2.2 The proposed context-aware deployment in $ABAC_{sh}$

As per the related work investigated in Section 2.1, we can conclude that to deploy an efficient context-ware feature, the attributes should be related to the system environment. Context-aware will add a flexibility to dynamic systems where the users and privileges keep changing such as the case in IaaS. ABAC is the basic access control type which can support the context-awareness. Therefore, we are not recommending RBAX extensions.

The proposed **$ABAC_{sh}$** model is adding context-aware through two phases. The first phase defines the context-attribute set. Each context-attribute consists of an attribute name and an attribute value. The context attributes-names set is predefined by the system administrator based on system critical information and characteristics. Context-attributes differ from the environment attributes in that the latter values are predefined by the administrator, whereas context-attribute values are updated based on the system states, where an embedded sensor captures the context information. For example, for the context-aware attribute named memory, its value will be updated based on the system memory measurements. The context attribute can reflect CPU clock, desk space, network zone, or data and time. In the second phase, context-awareness will be defined as one of the configuration points in the proposed **$ABAC_{sh}$** system to enforce the use of context in the access-control decision.

2.3 Critical analysis of SoD in ABAC

In an environment that allows policy combination, a user is authorized to act in more than one role or trigger more than one operation simultaneously. Policy combination might lead to policy conflict, as some actions violate the overall policy if they are committed at the same time. Therefore, constraints should be configured to manage this possibility.

The Separation of Duty (SoD) principle is used in such scenarios to prevent misuse of the system by limiting the user to the least privilege necessary to perform their required tasks. The least privilege principle limits the access of the subject during an operation on a specific task to be within the minimum resources, lowest privileges, and specified period of time. Several security enhancements can be gained from SoD, such as fraud prevention and error minimization [14–16].

There are two main types of SoD: static, and dynamic. Static-SoD (SSoD) will list the conflicting roles which cannot be executed by the same user at the same time, whereas dynamic-SoD (DSoD) enforces the control at the time of

access-request. In an RBAC model, roles and role relations are defined in advance during the policy engineering process. For SSoD in RBAC, SSoD relations place constraints on the user-to-rule assignment function, where one user can be assigned a specific set of roles and be excluded from another set of roles. Otherwise, two or more users are required to be involved in accomplishing sensitive tasks, since it is less likely that multiple parties will issue a fraud attack. In the DSoD relation, the capabilities for one user are restricted to being activated during a specific user session, i.e. the same user cannot perform two roles simultaneously [17, 18].

Although in RBAC, SSoD and DSoD relations offer some advancement in control over identity-based systems, security issues remain. The most accommodating form of SoD is History based (HSoD). Although, enforcing it in a static based access control management environment such as RBAC is di_cult, if not impossible [19, 20]. One role of a HSoD is that it prevents an object from being accessed by the same subject a certain number of times [21]. Therefore, we assume that the ABAC model concept has the characteristics of supporting certain types of SoD. The following section will investigate efforts in the literature to involve the SoD principle in ABAC.

A significant amount of research has been conducted regarding the principle of Separation of Duty (SoD) in RBAC; however, SoD deployment in ABAC remains a problem [22]. One of the earliest related works in specifying constraints in ABAC is illustrated through ABAC configuration-points [5]. Nevertheless, their proposed constraint settings are event-specific during attribute assignment and/or modification of the object and subject. This method is similar to the RBAC constraints-setting concept, where the allowed roles are activated for a specific user session after the roles are assigned to the users.

The author of ABCL proposed an event-independent constraint language based on conflicting relations of attribute values such as mutual exclusion and precondition [23]. ABCL language specifies restrictions either on a single set of attribute values or on a set of values of different attributes within the same entity. The usefulness of ABCL language has been validated through case studies. However, it lacks a framework or a formal model that illustrates its implementation. Dynamic Separation of Duties (DSoD) is more appropriate to cloud computing, and it also meets the dynamic nature of ABAC. Nguyen [24] has carried out interesting research on DSoD and proposed DSoD deployment through Provenance based Access Control (PBAC). His work is basically proposing a means to capture and utilize the information needed in the SoD enforcement, as previous work in the area assumes that the information is ready without demonstrating how to prepare it. Some of the previous work related to dynamic-based SoD is ObjDSoD, which is based on the object, and where the enforcement is constructed on a set consisting of conflicting-roles and a conicting-action on these roles. Therefore, the subject will not be allowed to perform an action on an object if that action is in the set of action role conflict. Another approach is OpsDSoD based on operations. This is a task-aware that involves an action-role conflict set, thus it differs from ObjDSoD by limiting the user to perform the needed actions for a particle task even though they have more privileges. A third approach is HDSoD, which combines ObjDSoD and OpsDSoD. Further, HDSoD is object-aware and a task aware. HDSoD is order-aware, where order-dependency conflict is triggered if the order is essential for a sequence of sub-tasks. Nguyen in [24] extended HDSoD by adding dependence-path-aware and past attribute-aware in their DSoD which is used in Provenance-based Access Control (PBAC).

Event pattern and response relations called obligations are introduced by Ferraiolo, Atluri, and Gavrila in their policy machine research [19], which can enforce some forms of HSoD in their access control framework. Obligations have

a set of conditions that are specified by the event pattern under which the state of the policy is obligated to change; only if this set matches the surrounding context, the operation on an object can be executed. There are two recognized standards can be applied to the ABAC concept: Extensible Access Control Markup Language (XACML), and Next Generation Access Control (NGAC) [25]. XACML does not show any support for DSoD constraint, while NGAC does show some support to DSoD through a Prohibitions (Denies)-relation, which includes a set of denying relations that specify privilege exceptions where a user that is allowed to run capability (x) will be prohibited from running capability (y).

2.4 SoD design and deployment in access control system

It is most likely that the formulation of SoD requirements are prepared by the administrator based on the business rules. An example of such a rule is, person may not approve his or her own purchase order [26]. SoD deployment can be involved with different layers of an access control system. It can be designed within administrative-level policies and procedures, or it can be used within logical or technical mechanism access-control restriction points [15].

Based on recommendations regarding SoD implementation to traverse its limitation in RBAC [20], several techniques have been explored, such as grouping concept, membership control, activation control, history control, and labels. However, in ABAC, the grouping concept will not be appropriate as grouping restricts an attributes flexible nature. Membership control cannot be adopted by ABAC as it is not role-centric. Though, the activation control concept has been adopted into SoD specifications in ABAC by Jin [5] and Bijon [23]. Ferraiolo et al. [19] describe a relation between entities that can be used in History based SoD deployment. Whereas Biswas et al. [27] point out that label concept can be used to enforce SoD in their proposed label-based access control in an ABAC. There are several obstacles in designing and implementing SoD, as it is an application-oriented policy where the business rules indicate the critical tasks which require SoD enforcement. Another challenge is that different applications may require various types of SoD. Lastly, most SoD types are informally defined, which creates ambiguity regarding the subjects or specifications [28].

2.5 The proposed SoD deployment in IaaS by $ABAC_{sh}$

Based on the above investigation [22–24, 29], SoD can be defined as an enforcement constraint configured to avoid conflict between policies. This conflict can be due to multi-access requests from different subjects to the same resource simultaneously, or the same subject requesting access to multiple resources at the same time. From this definition, it can be observed that SoD may be viewed as object-operation-oriented, which can be aligned with ABAC's relation between appropriate to enhance SoD by implementing a form of HSoD which will be suitable to be enforced in a dynamic access control policy environment such as ABAC. With RBAC, the centric entity involved in the SoD principle design is the role set. In contrast, ABAC cannot consider a role in the form of an attribute as it can lead to a chaos [30]. Therefore, the focus of this paper regarding formally defining SoD within ABAC will be on attributes and attribute-relations, with no aim to define an application-oriented SoD. Thus, we aim to identify a logical based design for SoD within the ABAC policy model. The proposed work is based on formal logic; exception cases are not encouraged in a formal logic as exceptions make regulations non-monotonic and introduce conflict between proven conclusions [31].

Therefore, the proposed SoD is operation-object orientated that defines a rules-set reflecting the forbidden operations on the set of objects under a specific situation of a collection of entities attributes. Entities include the object, the subject, the environment, and the system context. Moreover, formal logic facilitates SoD rule creation, even by non-expert security administrators. Since the proposed system is attribute-based, it is not necessary to update different locations if a new action restriction is added, deleted, or modified. Object-attributes and operations. We can discern from the above that it is more appropriate to enhance SoD by implementing a form of HSoD which will be suitable to be enforced in a dynamic access control policy environment such as ABAC. With RBAC, the centric entity involved in the SoD principle design is the role set. In contrast, ABAC cannot consider a role in the form of an attribute as it can lead to a chaos [30]. Therefore, the focus of this paper regarding formally defining SoD within ABAC will be on attributes and attribute-relations, with no aim to define an application-oriented SoD. Thus, we aim to identify a logical based design for SoD within the ABAC policy model. The proposed work is based on formal logic; exception cases are not encouraged in a formal logic as exceptions make regulations non-monotonic and introduce conflict between proven conclusions [31]. Therefore, the proposed SoD is operation-object orientated that defines a rules-set rejecting the forbidden operations on the set of objects under a specific situation of a collection of entities attributes. Entities include the object, the subject, the environment, and the system context. Moreover, formal logic facilitates SoD rule creation, even by non-expert security administrators. Since the proposed system is attribute-based, it is not necessary to update different locations if a new action restriction is added, deleted, or modified.

3. An intelligent framework for $ABAC_{sh}$

The framework is designed based on knowledge-agent and it employs rule-based expert system method. This intelligent system is not based on machine learning which will have a percentage of correct answers. This system is based on the available rules; therefore, it is not a type of uncertain approach. The system must guarantee an access decision.

The purpose of this **$ABAC_{sh}$** framework is to prove that AI architecture can contribute in supporting a dynamic access control. In regard to guaranteeing behavior, the followed mechanism in this chapter is based on knowledge available. If there is a shortage in knowledge, the access decision will be denied. There are other AI categories related to uncertain knowledge such as probabilistic reasoning, However, uncertain reasoning is out of this research scope.

3.1 AI scope for the proposed framework

According to [32–34], artificial intelligence systems are designed to think and act. They can be categorized into four types based on the intention of the system: Thinking Humanly, Acting Humanly, Thinking Rationally and Acting Rationally. The category of Thinking Rationally leads to an evolved need for the logic field in artificial intelligence. Involving logic in an intelligent system faces two substantial obstacles. The first one is the difficulty of presenting informal-knowledge using a formal logical notation though the certainty level is less than 100%. The second is that solving problems theoretically is different from solving them practically when the machine capacity is taken into consideration.

The category of Acting Rationally initiates the development of a computer agent. Prior to computer science, the term agent was used in different fields.

Therefore, there are various definitions of agent. However, it can be defined as an entity that acts within an environment by sensing its surroundings to update its knowledge and acts upon that to meet specific goals [35]. The agent function represents an abstract mathematical description, whereas the agent program represents an agent implementation within a physical system.

Problem-solving through an intelligent agent involves four stages. Firstly, the agent formulates its goal. Secondly, it formulates the problem based on five steps: initial state, possible actions, transition model that describes what each action does, goal test and path cost. Thirdly, it searches for a solution by looking for a sequence of actions that leads to the goal. Fourthly, in the execution stage, the solution found is implemented. However, the problem-solving agent is inflexible as each possible state should be hard-coded. Therefore, the complexity of the search stage grows exponentially in relation to the number of states in addition to its inability to infer unobserved information. Therefore, there is a need for logic to reason about the possible states instead of hard-coded all predicted states.

Knowledge-based reasoning is a step in overcoming problem-solving agent limitations. The logic provides a natural language for describing and reasoning about the system. The knowledge-based system is given facts about the external world, and it is asked queries about that world. The rule-based expert system is a popular method that is used to build knowledge-based systems. The rules are used to represent knowledge in the format of IF-THEN. The Inference engine is the reasoning component whereby the system concludes by linking the rules given in the knowledge base with facts supplied from the database. The explanation facilities allow the user to interact with the expert system to get justifications regarding the results produced by the inference engine. Therefore; the AI scope for the proposed intelligent-framework for $\mathbf{ABAC_{sh}}$ is illustrated in **Figure 1**. Modal logic is found to be the most appropriate logic to be used in AI as discussed by [36].

3.2 Logical-based agent architecture

Intelligence security is a fertile approach, as most existing security paradigms suffer from reactive and fragmented approaches [37]. In a frequently changing infrastructure, deploying an agent-based mechanism will be an advantage [38].

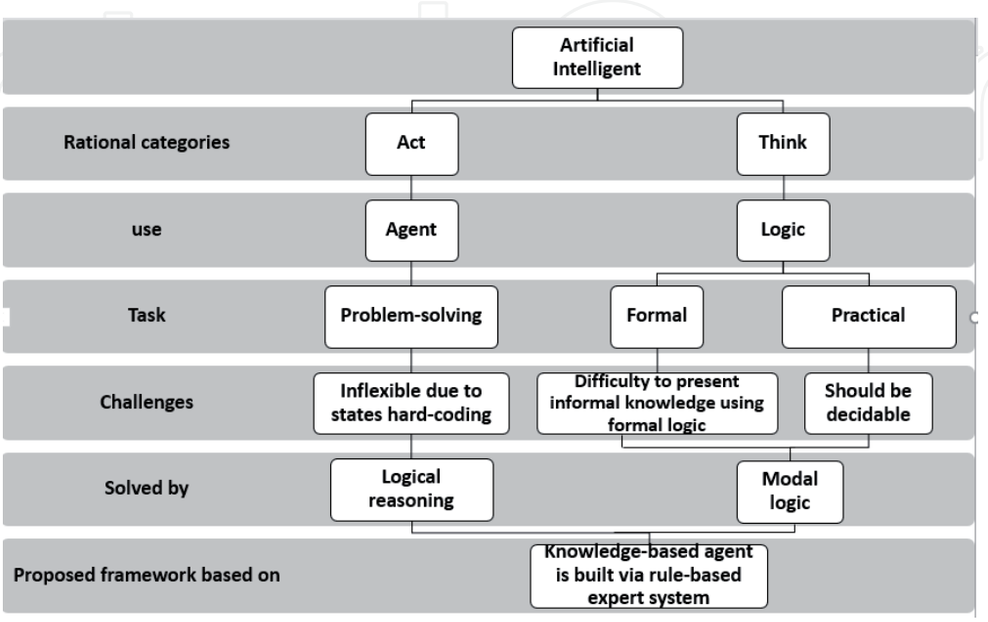

Figure 1.
AI scope for the proposed framework.

Modal logic is a candidate that supports a logical approach in artificial intelligence systems [39]. The main component of a knowledge-agent is a Knowledge-Base (KB) that consists of a set of sentences expressed using formal logic, in addition to two generic functions that involve logical inference. The first function is known as TELL, and adds new sentences (facts) to the KB to provide it with the required information. The second function is known as ASK, and queries the known information from the KB to determine the next step. The process between TELL and ASK will end as soon as the desired action is selected. The interaction between these two generic functions is similar to the updating and querying in databases, as illustrated in **Figure 2**. When an agent program is called upon, it performs two main actions. Firstly, it will TELL the KB what it perceives. Secondly, it ASKs the KB what action should be taken.

Therefore, agent-based architecture is suitable to represent an ABAC model. The logical agent, furthermore, will be appropriate for the proposed modal logic scheme. **Table 1** demonstrates how knowledge-based agent architecture can represent an ABAC system. The logical agent can be designed to represent an access-request state through a process of inference to derive a new representation of the access-request state that can be used to deduce required actions. The proposed access-control logic agent will be founded on knowledge-based agents, as this type of agent is logic-based [34].

3.3 $ABAC_{sh}$ conceptual requirement

Based on the analysis and investigations addressed an analytical study published by this chapter author in [40], the critical requirements in designing an ABAC model are listed below.

- Req.1 ABAC model definition requires to identify the configuration points. Each point should be formalized via the proper languages. The configuration point indicates the necessary configurations to be accomplished via the ABAC model processing for computing the access decision. These points are known as functional points. It is more convenient to minimize the number of configuration points as they affect the system's computational complexity.

- Req.2 ABAC is identity-free. Therefore, identifications such as subject-id are not the main elements in access-decision processing.

- Req.3 Avoid the creation of lists or groups in the design, as ABAC is intended to be flexible and able to cope with large enterprises.

- Req.4 Context-attributes reflect the current system state, whereas environment attributes reflect the fixed system characteristics.

- Req.5 ABAC is a multi-factor decision. Therefore, it enables fine-grained access

- Req.6 There are no predefined privileges for subjects as the privileges are computed after an access request is triggered. Policy rules set in ABAC are specified based on attributes. As a result, the permissible operations will be defined upon access-request.

- Req.7 The two basic functionalities in ABAC are attribute-assignment and rules-creation.

- Req.8 Security principles such as Separation of Duty (SoD) must be enforced.

```
function KB-AGENT(percept) return an action
static: KB, a knowledge base
t, a counter, initially 0, indicating time
TELL(KB,MAKE-PERCEPT-SENTENCE(percept,t))
action←ASK(KB,MAKE-ACTION-QUERY(t))
TELL(KB, MAKE-ACTION-SENTENCE(action,t)
t←t + 1 return action
```

Figure 2.
A generic knowledge-based agent function [34].

Components	Agent architecture	ABAC requirements
knowledge base(KB)	Background sentences	Predefined sets of entities, attributes and policy rules
	To represent action(s)	The action is access-decision
inference system	Infer (i.e arrive to a conclusion via reasoning) hidden properties of the world to add new sentence to KB	New sentences are added each time an access-request is triggered which consist of a combination of attributes with the request operation
	Infer based on the predefined sentences and the new ones to conclude with appropriate actions	Reasoning based on the attributes set and the policy rule-sets to conclude with an appropriate action (allow or deny) the access-request

Table 1.
Mapping knowledge-based agent with ABAC requirements.

The enhanced attribute-based access control **$ABAC_{sh}$** fulfills requirement Req.1 by employing one main configuration point that is ABAC agent. This agent takes as an input, the access request parameters which consist of the subject, the object, and the operation (s, o, opr). Then it returns the access decision that indicates if the subject is allowed to operate on the object or it is denied. Compared to ABAC α , which has four configuration points, the policy configuration here is reduced to one, as the proliferation of policy configuration points can introduce difficulties in policy expression and comprehension [5]. For Req.2, in the Policy Decision Point (PDP), the decision-making process considers the subject attributes in addition to other attributes, instead of depending solely on the subject identity information. In Req.3, grouping is studied by HGABAC [41] to facilitate the addition of a hierarchy feature to ABAC.

However, grouping and listing will impede the flexible nature of ABAC [30, 42]. Therefore, permissions grouping and listing are avoided in this **$ABAC_{sh}$**. The decision calculation is based on four sets of attributes: subject-attributes, object-attributes, environment-attributes, and system-context attributes, all of which are taken into consideration in the proposed design to meet requirement Req.4. System context attributes have a special sensor to obtain an up to date system state to meet requirement Req.5. The privilege decision is calculated based on the attributes relation defined in the policy-rules. Therefore; the privilege value is returned after the access-decision is triggered, which meets the requirement Req.6. There are two core functions of **$ABAC_{sh}$**. The first function takes place at the initial system stage, where the attribute pairs (name:value) are created for the defined access control system entities

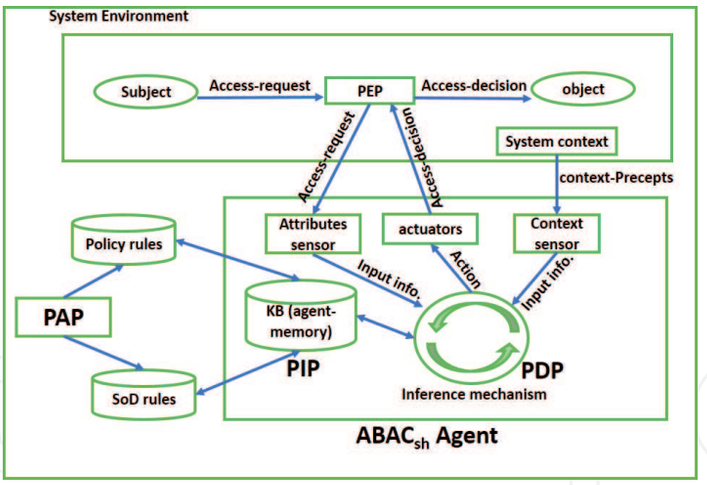

Figure 3.
The proposed intelligent framework for ABAC$_{sh}$.

(subject, object, environment, and context). The second function is rules creation, which represents the SoD-rules and Policy-rules in the form of capability which indicates the access-rights. These two functions meet the ABAC requirement Req.7. An initial SoD is introduced in this design in the type of a DSoD. The elimination of policy-rule conflicts can be achieved by an object-operation oriented constraint. A formal presentation of the proposed SoD enforcement sentences is defined and will be flexible to manage the set of constraints and meet the system requirements Req.8 since the administrator can modify the set of SoD sentences. The proposed SoD will be enforced after the access-request is triggered where an action is forbidden based on a collection of attributes.

3.4 The proposed framework

The proposed intelligent framework for **ABAC$_{sh}$** has been published by this chapter author in [40]. **Figure 3** shows the framework components. The proposed **ABAC$_{sh}$** model framework focuses on three functional points in Ref. to XACML framework: PDP, PIP and PAP. The Policy Enforcement Point (PEP) enforces the access decision. The PDP (Policy Decision Point) involves the core logical reasoning that takes place in an inference mechanism where the access decision is processed. The PAP (Policy Administration Point) involves rule creation by the system administrators. The PIP (Policy Information Point) involves information collection.

4. ABAC$_{sh}$ implementation for IaaS cloud via OpenStack

This section demonstrates the visibility of **ABAC$_{sh}$** in IaaS cloud by introducing an enforcement architecture based on OpenStack. That is followed by a prototype implementation and performance evaluation that illustrates the advantages of the proposed **ABAC$_{sh}$** extension over the existing access control model. This section discuss the following points

- Designing enforcement architecture for **ABAC$_{sh}$** that utilizes telemeter service deployment to be used in feeding Policy Information Point (PIP) with attributes values.

- A prototype implementation of an extended nova access control model with an intelligent ABAC.
    - Extend the nova policy enforcement point (PEP) to communicate with an external policy engine
    - The proposed external policy-engine works as policy decision point (PDP)
- The introduced PDP follows **$ABAC_{sh}$** by
    - Utilizing the attributes in access decision-making process.
    - Involving forward-chaining algorithm that works as logical reasoning for access decision processing.
- Three experiments are studied to compare and contrast the extended **$ABAC_{sh}$** with the default nova-OpenStack access control model.
- The Quality of Service (QoS) measurement is discussed based on response time as a performance metric.

5. OpenStack access control model (OSAC)

A key component in building a virtualization environment is its operation via the hypervisor. The hypervisor on its own cannot build IaaS. Therefore, a cloudstack such as OpenStack, Cloud-Stack or OpenNebul is required. According to the current industry, OpenStack is likely to become a dominant cloud-stack [43]. OpenStack is an open-source cloud computing platform that offers an IaaS layer of service. OpenStack IaaS infrastructure supports agent communication. For example, network nodes in the OpenStack activate a DHCP agent to deploy a DHCP service [44]. OpenStack was selected to be the experimental platform for this research as it has a supportive and active community of both academic researchers and commercial bodies.

OpenStack can deploy different access control models within its infrastructure [45]. For example, nova configuration files can be protected via several implementations such as centralized logging, policy file (policy.json) and MAC framework (Mandatory Access Control). The availability of access control models depends on the hypervisor vendor. The supported models are Mandatory Access Control (MAC), Discretionary Access Control (DAC), and Role-Based Access Control (RBAC).

The Openstack access control model (OSAC) that enables both operators and users to access resources for specific services is a type of RBAC [46]. The keystone [47] supports the notation of roles and groups. Each user should be associated with a group, and each group has a list of roles. For a user to be granted access to a service, Openstack service takes into consideration his/her role, though as the first authorization step, the OpenStack PEP (Policy Enforcement Point) takes into consideration the policy rules associated with the resources before it checks the user role. Therefore, the policy enforcement middleware enables fine-grained access. Each Openstack service defines the access control policies rules for its resources in a specific policy file called policy.json.

5.1 Policy engine

Policy engine in OpenStack is a type of authorization engine that return back a decision based on some policy rules that indicating if a specific operation is allowed or denied [14, 45, 48]. The default policy engine is maintained via Oslo policy, and the access request is issued via API communication. Oslo policy is completely separated from RBAC model [49]. The developer can view Oslo policy rules that are related to nova via the command "oslopolicy-policy-generator {namespace nova". The list of rules verifies if the user credentials are matching to grant access to the requested resources. The user credentials are stored in the format of a token. The token holds information related to the token itself in addition to the user, the project, the domain, the role, the service and the endpoint. The policy rules are stored in JSON (JavaScript Object Notation) file format.

In policy.json, the access policy consists of two main parts "<target>": "<rule>" [47] . The target is known as an action that indicates the API call for an operation such as "start an instance". The rules can be one of the following: allow all, deny all, a special check, a comparison of two values, Boolean expressions based on simpler rules. The special check gives the developers an opportunity to extend the OpenStack policy engine. The special check can indicate, a role that is extracted from token information, or a complex rule by defining an alias, or a target URL that delegates the check to an external policy engine.

5.2 Nova authorisation data-flow

Each service in Openstack has its own access control configuration points which involve PEP, PIP, PDP and PAP. The information ow between nova access-control configuration points is demonstrated in **Figure 4**. In the original Openstack architecture, Nova PEP will send a token that contains the information of the access request to Nova PIP to retrieve the object information. Then Nova PEP sends the information of the subject, object and request to Nova PDP in order for Nova PDP request an access control policy from Nova PAP. Nova PDP evaluate the access request based on the policy and return the access decision to Nova PEP.

5.3 Forward-chaining algorithm

In the search stage of the problem-solving agent, there is a need to use a proper searching algorithm that meets the problem scenario and the input information. The search algorithms that are used in rule-based systems are backward chaining, forward-chaining and a mixture of both of them [34, 51]. **Table 2** compares between the reasoning algorithms which are referred to as chaining in some literature.

Many researchers avoid the Logic Theory Machine, which is based on forward-reasoning due to the computation complexity. However, this complexity is due to the classical mathematical logic and it is not due to the forward-reasoning concept [52]. Classical mathematical logic such as propositional logic and First-order logic. Therefore, the computational complexity of forward-chaining when it is used in nonclassical logic such as deontic logic will be decidable, and it will have an acceptable computation complexity. A simple algorithm for forward-chaining is illustrated in **Figure 5**. Forward reasoning search iteration is based on facts and rules to find a conclusion.

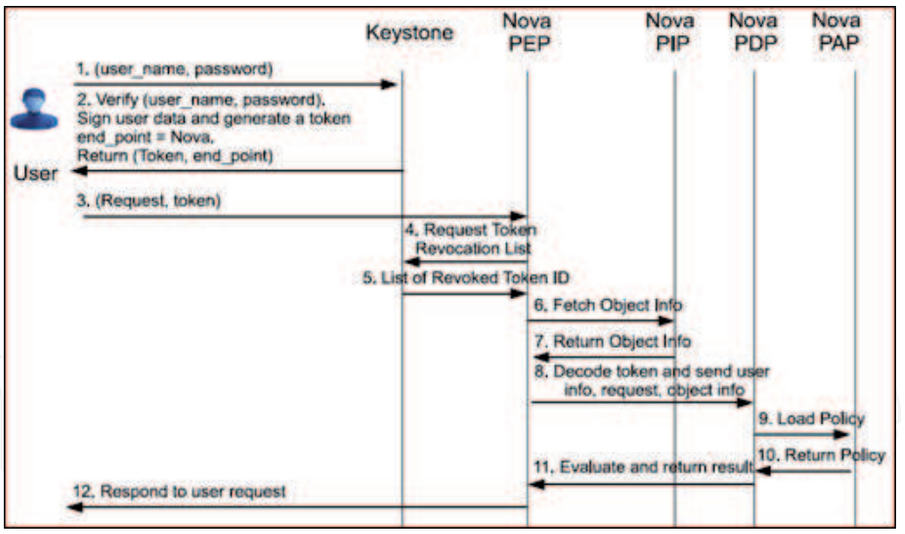

Figure 4.
Nova authorization data- flow [50].

	Forward-chaining	Backward-chaining
Known as	Forward reasoning (Data driven)	Backward reasoning (Goal driven)
Reasoning start with	A Set of facts to reach a goal (or hypothesis)	A hypothesis (goal) to reach the facts behind it
When applicable	If the goal is unknown	If the set of goals are known

Table 2.
Comparing forward-chaining with backward-chaining.

```
newFacts=False
For rule in rule-list
     if all premises match fact-base:
          For each fact in consequences:
               if fact not in fact-base:
                    add fact to fact-base
                    new Facts = True
If newFacts: goto 1
```

Figure 5.
Forward chaining algorithm.

5.4 ABAC implementation in OpenStack

5.4.1 Enforcement architecture

The core characteristics of **$ABAC_{sh}$** are to work as an intelligent agent that sense the attributes (the environment, the system context, the subject and the object) in order to search for an access decision using forward chaining (forward-reasoning). The set of attributes represent facts whereas the set of policy rules represent the rules.

The proposed **ABAC$_{sh}$** enforcement architecture employs the Telemetry service of OpenStack. The telemetry service in Openstack provides the facility to sense the IaaS cloud for environment attributes and the context attributes. The Telemetry service facilitates polling information from the computing service since the proposed access control agent **ABAC$_{sh}$** will use the collected information for the attributes assignment process. As an example, the nova service access control process will be used to illustrate **ABAC$_{sh}$** extension. Section 4.3 introduces the default data ow of nova service access control while **Figure 6** illustrates nova service access control with **ABAC$_{sh}$** extension. The proposed **ABAC$_{sh}$** enforcement architecture focuses on three configuration points: PIP (Policy Information Point), PAP (Policy Admission Point) and PDP (Policy Decision Point). **ABAC$_{sh}$** enforcement architecture is divided into three components as follows:

1. **ABAC$_{sh}$** PIP: this is used to collect attributes information from the access control entities, the environment, and the system context. PIP can be achieved in Openstack through configuring the Telemetry component. The Telemetry service is designed to support a billing system by gathering the required information. Therefore, its structure will be beneficial in providing PIP with required attribute information. Telemetry consists of five building blocks: Compute Agent, Central Agent, Collector, Data Store and API Server in order to perform five essential functions [53]. **Figure 7** summarizes the Telemeter process to collect data for further analysis. Telemeter can be configured to collect the attributes and save them in JSON file as this file format is used to store policy rules in OpenStack

2. **ABAC$_{sh}$** PAP: The knowledge database for **ABAC$_{sh}$** model consists of access rules from SoD rules and Policy rules. The access rules are created by the system administrator. Those rules will be stored in JSON file format to facilitate its implementation in OpenStack.

3. **ABAC$_{sh}$** PDP: this is the logical component which reasons about access control in **ABAC$_{sh}$**. **ABAC$_{sh}$** PDP will get an access-request sentence from **ABAC$_{sh}$** PEP that consist of the attributes information with the access request. **ABAC$_{sh}$** PDP will load the access rules from **ABAC$_{sh}$** PAP. **ABAC$_{sh}$** PDP accomplishes logical reasoning through forward-chaining algorithm. The result of the logical reasoning indicates if the access is permitted or denied.

5.4.2 Prototype implementation

The first stage of **ABAC$_{sh}$** deployment in Openstack is to be implemented on nova component. **ABAC$_{sh}$** PDP part will be implemented as a prototype.

- **Scope and Assumption.**

  IaaS access control tenant scope can be a single tenant [5], multi-tenant [54–56] and collaborating parties a cross-clouds [57, 58]. The implementation scope of access control in this chapter is within single tenant whereas its hypothesis is applicable to multi-tenant and cross-clouds as the big concept behind **ABAC$_{sh}$** is user-id free and attributes-based. The proposed **ABAC$_{sh}$** is not replacing OpenStack RBAC in this stage. Instead, it allows fine-grained access control and opens prospective avenues to replace RBAC in the near feature.

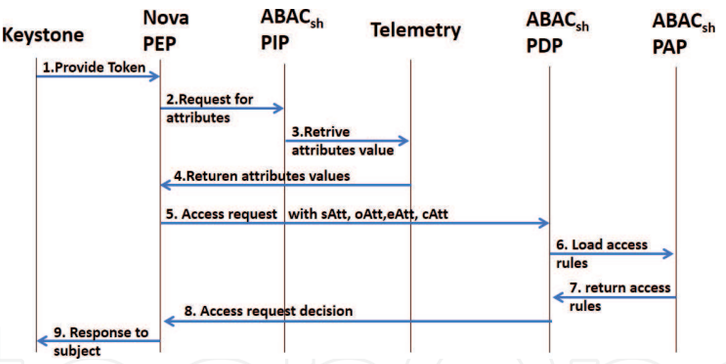

Figure 6.
Proposed $ABAC_{sh}$ for Openstack nova.

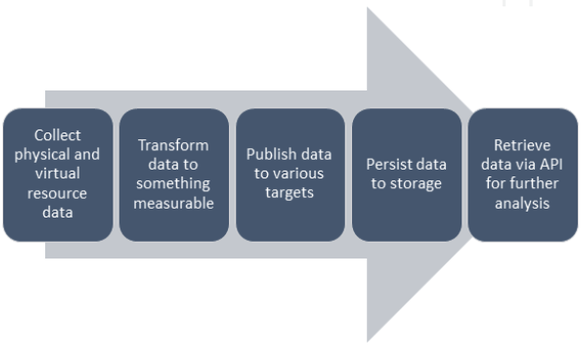

Figure 7.
Telemeter process.

- **OpenStack Testbed**.

  OpenStack aids in deploying IaaS cloud. **Figure 8** shows the deployed testbed in this chapter. It is installed in three machines using Ubuntu 16.04 LTS as an operating system and OpenStack Ocata the latest release (Feb2017). One machine is configured as a controller which provides OpenStack main server in addition to networking services (neutron), keystone, nova and glance. The Two other machines are configured as compute nodes where virtual machines are hosted. The machines specification is Intel Core i5–4460 CPU Processor 3.20GHz _ 4, 15.5 GiB memory, 235 GB Disk and 2 NICs cards. The testbed networking consists of two LANs: management network and data network. The management network traffics the Openstack service communication where data network connects the communication of the virtual machine. This IaaS is a private cloud where OpenStack services and the VMs are accessed by the LAN users.

- **Data flow**.

  Nova policy engine is embedded within its configuration files, therefore it is considered as one of OpenStack's limitations. However, the default policies can be overwritten if policy.json is enabled. Policy.json can be configured to call an external policy engine through URL. The token hold information that can be passed from OpenStack keystone to **$ABAC_{sh}$** policy engine via RESET GET-call. Nova PEP receives an access decision from **$ABAC_{sh}$** policy engine via RESET POST-call. **$ABAC_{sh}$** policy engine use a forward-chaining algorithm to produce an access control decision. The access control reasoning takes facts which are subject and object attributes, in addition to the system and context attributes.

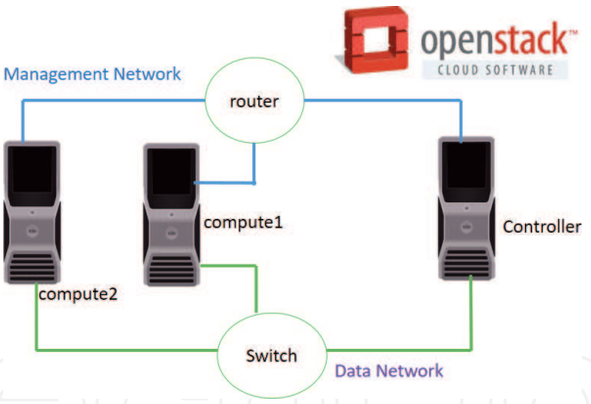

Figure 8.
OpenStack testbed in physical machines.

Based on access rules defined by the administrator, the access request will be allowed or denied. **Figure 9** shows the **ABAC$_{sh}$** PDP extension to nova authorization. A policy engine is designed and implemented to add an access control enforcement based on attributes.

In access control terminology, the Openstack users are the subject, the nova resources are the objects, the policy.json is PEP and **ABAC$_{sh}$** policy engine is the PDP. The attributes are extracted from the following channels

- The subject attributes can be extracted from keystone token where the available information is user name, user id, user password, role id, role name. The Token information can be retrieved from "content-type:application/json' 'through curl command.

- The extracted nova metrics from the OpenStack system via a command openstack quota show are considered as the object attributes. The attributes information is stored in JSON _le format

- The nova environment attributes are extracted via the OpenStack command openstack hypervisor stats show. The attributes information is stored in JSON _le format

- The context attributes are not implemented in this prototype but it is visible to be included via Telemetry OpenStack service

ABAC$_{sh}$ policy engine server is implemented using several programming technologies. The web server is developed using Python programming language with web.py since OpenStack services is using python. RESETful API utilities are used to allow the communication between **ABAC$_{sh}$** and OpenStack APIs. The forward reasoning function is programmed using java since this programming language can be smoothly integrated into web programming. To allow the technical interaction between python and java, jpype is used [59, 60]. Data is stored in JSON file formats such as policy data and attributes data.

5.4.3 Performance evaluation

The aim of this performance evaluation is to detect if **ABAC$_{sh}$** deployment in Openstack introduces any significant overhead. The efficiency of deploying an access control model depends on several factors. The quality of service (QoS) measures

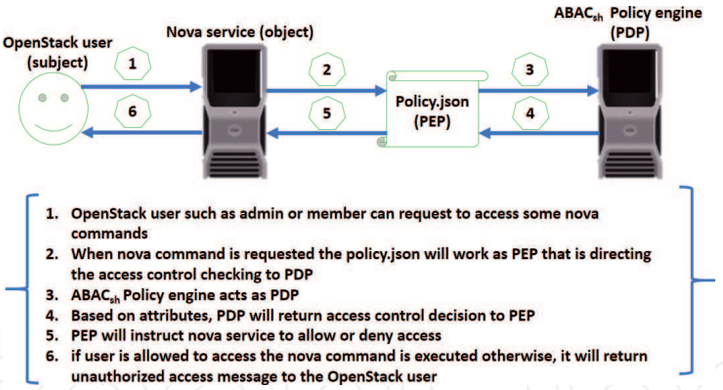

Figure 9.
The prototype implementation data flow.

can be calculated by performance properties and computation complexity [61, 62]. In this section, the performance metrics are evaluated. The performance metrics consist of four elements: response time, policy repository and retrieval, policy distribution, and Integrated with authentication function [62]. **Table 3** explains theses performance metrics elements and on which access control components they can be applied. Since the implemented prototype is the **ABAC$_{sh}$** PDP, the followed experiments will measure response time. With regard to policy repository and retrieval, the implemented **ABAC$_{sh}$** use JSON file to store access control policy which does not add any extra hardware or software cost to OpenStack IaaS cloud as this form of policy storage is used by OpenStack. The remaining two performance metrics elements are not calculated in this stage as PIP is not implemented.

5.4.4 Experiment content

In this section, the performance evaluation of the implemented **ABAC$_{sh}$** prototype in OpenStack is presented. Specifically, **ABAC$_{sh}$** policy engine which represents PDP of access control model is discussed. The experiments fall into three parts where the response time is calculated. Response time indicates the time consumed by the system in order to process the access request decision call. The response time has been used to measure the performance in several OpenStack implementations such as in [63, 64]. In these experiments, OpenStack cloud was installed in physical servers running Ubuntu 16.04 LTS release. Three types of execution time can be measured [65, 66]. The first one is real time that reflects the wall clock where the time is calculated from the start till the end of the call including the waiting time and time used by other processes. The second one is user-time that reflects the actual CPU-time spent outside the kernel during the process call in user-mode without considering other processes. The third one is sys-time that reflects the actual CPU-time spent within the kernel during the process execution.

Three experimental settings have been implemented as explained below.

- Experiment 1 (Exp1): The response time for the default access control model to process access request to nova resources. The default use RBAC and Oslo policy engine.

- Experiment 2 (Exp2): The response time for extending the default nova policy engine with **ABAC$_{sh}$** that utilizes 24 attributes in access control processing

- Experiment 3 (Exp3): The response time for extending the default nova policy engine with **ABAC$_{sh}$** that use forward-chaining for access control reasoning.

5.4.5 Experimental results and discussion

Each experiment was run five times, and then the average value was recorded. Five scenarios were observed by increasing the number of requests from five to twenty-five as illustrated in **Table 4**. The request indicates an access control request from a user (subject) to access nova-resources (object).

Based on usability engineering [67, 68] The response time value can be within three categories: over 0.1 seconds will give the user a feeling that the system is reacting instantaneously, over 1.0 seconds will give the user a feeling of a delay but

Performance metrics element	Description	The applicable Access Control Component
Response time	The time required to process access request should meet the organization requirement	PEP, PDP, PIP, PRP
Policy distribution	If there exist a mechanism that can be used for access control policy distribution	PAP, PIP, PRP
Integrated with authentication function	If the subject and object can be associated with some identifications through an authentication function.	PIP
Key: Policy Decision Point (PDP), Policy Administration Point (PAP), Policy Enforcement Point (PEP), Policy Information Point (PIP), Policy Retrieval Point (PRP)		

Table 3.
Performance evaluation metrics.

No. of Requests	Response time	Exp1	Exp2	Exp3
5	Real	8.67	8.96	8.77
	User	5.56	5.52	5.55
	System	0.42	0.44	0.40
10	Real	17.24	17.34	17.24
	User	10.99	11.05	11.11
	System	0.88	0.86	0.83
15	Real	25.52	26.08	25.87
	User	16.46	16.61	16.64
	System	1.25	1.31	1.22
20	Real	34.40	34.56	34.45
	User	22.07	22.17	22.23
	System	1.68	1.66	1.56
25	Real	43.02	43.05	43.07
	User	27.64	27.63	27.82
	System	2.09	2.10	1.96

Table 4.
Experimental results.

will stay uninterrupted, over 10 seconds the user will lose his/her attention and will search for something to work on till the computer responds.

Three time values has been recorded as illustrated in **Table 4**: real-time, user-time and sys-time. In this study, real-time and sys-time have a direct reflect on the performance analysis whereas user-time is reflecting the processing outside the kernel. The real-time shows the access control execution time in additions to the other OpenStack cloud processes that introduce some delay by blocking the process or introducing a waiting time. Therefore, this measurement will indicate the effect of our extended **$ABAC_{sh}$** nova to the overall OpenStack system.

The graph in **Figure 10** compares the real-time for the three set of experiments. The increase is 0.05 seconds when the extended **$ABAC_{sh}$** nova employ forward reasoning in access decision processing as shown in **Table 5** while the increase is 0.145 seconds when **$ABAC_{sh}$** uses twenty-four attributes in access decision processing. Therefore, there is an increase of 0.56% when attributes are added to the policy engine and 0.19% when the forward-chaining algorithm is added. Consequently, the increase in response time is negligible in Ref. to the usability engineering when the nova default access control is extended with part of the proposed ABAC enhancement.

On the other hand, sys-time gives the process execution only within the kernel regardless of the other tasks. Therefore, the time for the 25 requests dropped from 43.02 seconds within real-time to 2.09 seconds within sys-time during Exp1 which involve default nova access control. The sys-time comparison for the three experiments is illustrated in **Figure 11**. The results show a slightly better performance of

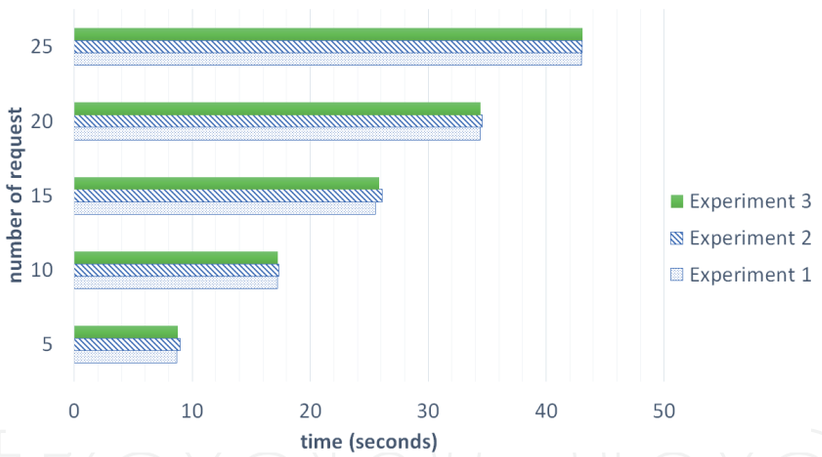

Figure 10.
Real-time for access control processing in nova.

Experiment 2 - Experiment 1	Experiment 3 - Experiment 1
0.29	0.1
0.1	0
0.56	0.35
0.16	0.05
0.03	0.05
Average	Average
0.145	0.05
Percentage	Percentage
0.56%	0.19%

Table 5.
Comparing real-time values.

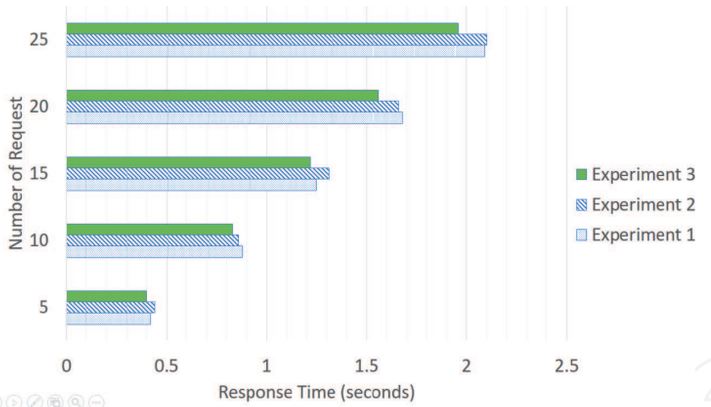

Figure 11.
Sys-time for access control processing in nova.

Experiment 2 - Experiment 1	Experiment 3 - Experiment 1
0.02	−0.02
−0.02	−0.05
0.06	−0.03
−0.02	−0.12
0.01	−0.13
Average	Average
0.05	−0.07
Percentage	Percentage
4%	−5.5%

Table 6.
Comparing sys-time values.

5.5% for extending the default nova access control when forward-reasoning has been utilized whereas an increase of 4% over the default nova when 24-attributes are used in the policy-engine as illustrated in **Table 6**.

From these results, the **ABAC$_{sh}$** shows an acceptable performance compared to the default OpenStack access control within nova service. This section demonstrates the enhanced attribute-based access control **ABAC$_{sh}$** performance improvement when attributes and forward reasoning algorithm are employed. It has been noticed that the performance improvement is liner in **Figure 10** when only attributes are involved in access decision. Whereas in **Figure 11** when forward reasoning is involved, an improvement in performance has been noticed. This indicates an opportunity of enhancing the IaaS-cloud security when logical reasoning and AI mechanism are involved in access control system.

5.4.6 Experiments limitations

The main aim of the experiments in this chapter is to study the performance improvement when attribute-based access control model is introduced into IaaS cloud. The experiment scale is limited to a private cloud in a LAN set-up. Therefore, the network performance metrics has not been studied such as the latency and throughput. The implementation in this chapter does not involve the PIP component of the access control, therefore only a simple forward reasoning algorithm has been deployed without knowledge update component. The used database for knowledge is written manually whereas the system should use an automated information collection method if PIP is implemented. One subject is involved in

each experiment, therefore multi-access has not been investigated in this chapter. Multi-tenant study is a critical future work.

6. Conclusions

This Chapter is focusing on the problem of deploying access control in a dynamic environment. Access control is one of the information security principles where the system user access is controlled through an access policy. In the cyber-security world where systems and devices are distributed in different locations, there is a need to have an access control model that is able to cope with a dynamic environment where new users with different privileges are joining and leaving the system.

This chapter is proposing to deploy an enhanced version of attribute-based access-control named **$ABAC_{sh}$.** This model is deploying knowledge base category of AI. A proof of concept is implemented in the cloud computing environment to measure the performance and the visibility of such a deployment.

Author details

Shadha Mohamed Sulaiyam ALAmri
University of Technology and Applied Sciences, Muscat, Oman

*Address all correspondence to: shadha-alamri@hct.edu.om

References

[1] W. E. Forum, "Fourth Industrial Revolution Beacons of Technology and Innovation in Manufacturing," Geneva, 2019. [Online]. Available: http://www3.weforum.org/docs/WEF_4IR_Beacons_of_Technology_and_Innovation_in_Manufacturing_report_2019.pdf.

[2] J. Qiu, Z. Tian, C. Du, Q. Zuo, ... S. S.-I. I. of T., and U. 2020, "A survey on access control in the age of internet of things," *ieeexplore.ieee.org*, vol. 7, no. 6, pp. 4682-4696, 2020, doi: 10.1109/JIOT.2020.2969326.

[3] Y.-G. Kim, C.-J. Mon, D. Jeong, J.-O. Lee, C.-Y. Song, and D.-K. Baik, "Context-Aware Access Control Mechanism for Ubiquitous Applications," Springer Berlin Heidelberg, 2005, pp. 236-242.

[4] Y.-G. Kim and Y. Lee, "Context Information-based Application Access Control Model," in *Proceedings of the 10th International Conference on Ubiquitous Information Management and Communication - IMCOM '16*, 2016, pp. 1-5, doi: 10.1145/2857546.2857623.

[5] X. Jin, "Attribute-based access control models and implementation in cloud infrastructure as a service," THE UNIVERSITY OF TEXAS AT SAN ANTONIO, 2014.

[6] A. Cavoukian, M. Chibba, G. Williamson, and A. Ferguson, "The Importance of ABAC: Attribute-Based Access Control to Big Data: Privacy and Context," *Priv. Big Data Institute, Ryerson Univ. Toronto, Canada*, 2015, Accessed: May 17, 2016. [Online]. Available: http://www.ryerson.ca/content/dam/pbdi/Resources/The Importance of ABAC to Big Data 05-2015.pdf.

[7] F. Li, "Context-Aware Attribute-Based Techniques for Data Security and Access Control in Mobile Cloud Environment." Apr. 01, 2015.

[8] K. Nahrstedt and R. Campbell, "Security for Cloud Computing," 2012. Accessed: Oct. 08, 2015. [Online]. Available: https://illinois.edu/blog/files/695/66281/2737.pdf.

[9] X. Yao, X. Han, and X. Du, "A lightweight access control mechanism for mobile cloud computing," in *2014 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, Apr. 2014, pp. 380-385, doi: 10.1109/INFCOMW.2014.6849262.

[10] D. Al Kukhun, "Steps towards adaptive situation and context-aware access: a contribution to the extension of access control mechanisms within pervasive information systems," Research Institute in Computer Science of Toulouse, 2012.

[11] Z. Liu and J. Wang, "A fine-grained context-aware access control model for health care and life science linked data," Multimed. Tools Appl., vol. 75, no. 22, pp. 14263-14280, Jan. 2016, doi: 10.1007/s11042-016-3269-6.

[12] K. K. Venkatasubramanian, T. Mukherjee, and S. K. S. Gupta, "CAAC -- An Adaptive and Proactive Access Control Approach for Emergencies in Smart Infrastructures," ACM Trans. Auton. Adapt. Syst., vol. 8, no. 4, pp. 1-18, Jan. 2014, doi: 10.1145/2555614.

[13] C. Choi, J. Choi, and P. Kim, "Ontology-based access control model for security policy reasoning in cloud computing," J. Supercomput., vol. 67, no. 3, pp. 711-722, 2014, doi: 10.1007/S11227-013-0980-1.

[14] S. Bhatt, F. Patwa, and R. Sandhu, "An Attribute-Based Access Control Extension for OpenStack and Its Enforcement Utilizing the Policy Machine," in *2016 IEEE 2nd International Conference on Collaboration and Internet Computing (CIC)*,

Nov. 2016, pp. 37-45, doi: 10.1109/CIC.2016.019.

[15] R. L. Krutz and R. D. Vines, *Cloud security: a comprehensive guide to secure cloud computing*. Wiley Pub, 2010.

[16] W. Li, H. Wan, X. Ren, and S. Li, "A Refined RBAC Model for Cloud Computing," in *2012 IEEE/ACIS 11th International Conference on Computer and Information Science*, May 2012, pp. 43-48, doi: 10.1109/ICIS.2012.13.

[17] D. F. D. Ferraiolo, R. Sandhu, S. Gavrila, D. R. Kuhn, and R. Chandramouli, "Proposed NIST standard for role-based access control," ACM Trans. Inf. Syst. Secur., vol. 4, no. 3, pp. 224-274, Aug. 2001, doi: 10.1145/501978.501980.

[18] D. Nguyen, J. Park, and R. Sandhu, "A provenance-based access control model," in *2012 Tenth Annual International Conference on Privacy, Security and Trust*, Jul. 2012, pp. 137-144, doi: 10.1109/PST.2012.6297930.

[19] D. Ferraiolo, V. Atluri, and S. Gavrila, "The Policy Machine: A novel architecture and framework for access control policy specification and enforcement," J. Syst. Archit., vol. 57, no. 4, pp. 412-424, 2011, doi: 10.1016/j.sysarc.2010.04.005.

[20] R. T. Simon and M. E. Zurko, "Separation of duty in role-based environments," in *Proceedings 10th Computer Security Foundations Workshop*, 1997, pp. 183-194, doi: 10.1109/CSFW.1997.596811.

[21] C. T. Hu, D. F. Ferraiolo, and D. R. Kuhn, *Assessment of Access Control Systems*. US Department of Commerce, National Institute of Standards and Technology, 2006.

[22] D. Servos and S. L. Osborn, "Current Research and Open Problems in Attribute-Based Access Control," ACM Comput. Surv., vol. 49, no. 4, pp. 1-45, Jan. 2017, doi: 10.1145/3007204.

[23] K. Z. Bijon, "Constraints for attribute based access control with application in cloud IaaS," THE UNIVERSITY OF TEXAS AT SAN ANTONIO, 2015.

[24] D. Nguyen, "Provenance-based access control models," The University of Texas at San Antonio, 2014.

[25] D. Ferraiolo, R. Chandramouli, R. Kuhn, and V. Hu, "Extensible Access Control Markup Language (XACML) and Next Generation Access Control (NGAC)," in *Proceedings of the 2016 ACM International Workshop on Attribute Based Access Control - ABAC '16*, 2016, pp. 13-24, doi: 10.1145/2875491.2875496.

[26] R. A. Botha and J. H. P. Eloff, "Separation of duties for access control enforcement in workflow environments," IBM Syst. J., vol. 40, no. 3, pp. 666-682, 2001, doi: 10.1147/sj.403.0666.

[27] P. Biswas, R. Sandhu, and R. Krishnan, "Label-Based Access Control," in *Proceedings of the 2016 ACM International Workshop on Attribute Based Access Control - ABAC '16*, Mar. 2016, pp. 1-12, doi: 10.1145/2875491.2875498.

[28] V. D. Gligor, S. I. Gavrila, and D. Ferraiolo, "On the formal definition of separation-of-duty policies and their composition," in *Proceedings. 1998 IEEE Symposium on Security and Privacy (Cat. No.98CB36186)*, 1998, pp. 172-183, doi: 10.1109/SECPRI.1998.674833.

[29] S. Jha, S. Sural, V. Atluri, and J. Vaidya, "Enforcing Separation of Duty in Attribute Based Access Control Systems," in *Information Systems Security*, Springer, Cham, 2015, pp. 61-78.

[30] E. Coyne and T. R. Weil, "ABAC and RBAC: Scalable, Flexible, and Auditable Access Management," IT Prof., vol. 15, no. 3, pp. 14-16, May 2013, doi: 10.1109/MITP.2013.37.

[31] N. Dinesh, A. Joshi, I. Lee, and O. Sokolsky, "Permission to speak: A logic for access control and conformance," J. Log. Algebr. Program., vol. 80, no. 1, pp. 50-74, 2011, doi: 10.1016/j.jlap.2009.12.002.

[32] R. Fagin, J. Y. Halpern, Y. Moses, and M. Y. Vardi, *Reasoning about knowledge*. MIT Press, 2003.

[33] M. Negnevitsky, *Artificial intelligence: a guide to intelligent systems*, 3rd ed. Addison Wesley/Pearson, 2011.

[34] S. J. (Stuart J. Russell, P. Norvig, and E. Davis, *Artificial intelligence: a modern approach*. Prentice Hall, 2010.

[35] A. J. Soroka, "Agent-based System for Knowledge Acquisition and Management Within a Networked Enterprise," in *Artificial Intelligence Techniques for Networked Manufacturing Enterprises Management*, Springer London, 2010, pp. 43-86.

[36] R. Mastop, "Modal Logic for Artificial Intelligence," 2012, Accessed: Mar. 11, 2017. [Online]. Available: http://www.phil.uu.nl/~rumberg/infolai/Modal_Logic.pdf.

[37] R. Knights and E. Morris, "Move to intelligence-driven security," Netw. Secur., vol. 2015, no. 8, pp. 15-18, Aug. 2015, doi: 10.1016/S1353-4858(15)30071-4.

[38] F. Doelitzscher, C. Reich, M. Knahl, A. Passfall, and N. Clarke, "An agent based business aware incident detection system for cloud environments," J. Cloud Comput. Adv. Syst. Appl., vol. 1, no. 1, p. 9, Jul. 2012, doi: 10.1186/2192-113X-1-9.

[39] M. Huth and M. Ryan, *Logic in computer science: modelling and reasoning about systems*. Cambridge University Press, 2004.

[40] S. M. Sulaiyam Al Amri, "IaaS-cloud security enhancement: An intelligent attribute-based access control framework," in *2018 Majan International Conference (MIC)*, Mar. 2018, pp. 1-9, doi: 10.1109/MINTC.2018.8363159.

[41] D. Servos and S. L. Osborn, "HGABAC: Towards a Formal Model of Hierarchical Attribute-Based Access Control," in *Foundations and Practice of Security*, Springer, Cham, 2015, pp. 187-204.

[42] V. C. Hu *et al.*, "Guide to Attribute Based Access Control (ABAC) Definition and Considerations," 2013. Accessed: May 14, 2016. [Online]. Available: http://www.itbusinessedge.com/itdownloads/security/guide-to-attribute-based-access-control-abac-definition-and-considerations.html.

[43] W. Huang, A. Ganjali, B. H. Kim, S. Oh, and D. Lie, "The State of Public Infrastructure-as-a-Service Cloud Security," ACM Comput. Surv., vol. 47, no. 4, pp. 1-31, Jun. 2015, doi: 10.1145/2767181.

[44] OpenStack, "OpenStack Docs: Configure neutron agents," 2017. https://docs.openstack.org/admin-guide/networking-config-agents.html (accessed Apr. 02, 2017).

[45] OpenStack, "OpenStack Docs: OpenStack Security Guide," 2017. https://docs.openstack.org/security-guide/ (accessed Jul. 28, 2017).

[46] X. Wen, G. Gu, Q. Li, Y. Gao, and X. Zhang, "Comparison of open-source cloud management platforms: OpenStack and OpenNebula," in *2012 9th International Conference on Fuzzy Systems and Knowledge Discovery*

(FSKD), May 2012, pp. 2457-2461, doi: 10.1109/FSKD.2012.6234218.

[47] OpenStack.org, "OpenStack Docs: The policy.json file," 2017. https://docs.openstack.org/ocata/config-reference/policy-json-file.html (accessed Aug. 06, 2017).

[48] A. Young, "Dynamic Policy for Access Control," OpenStack Summit May 2015 Vancouver, 2015. https://openstacksummitmay2015vancouver.sched.com/event/2qcK/dynamic-policy-for-access-control (accessed Jul. 29, 2017).

[49] OpenStack.org, "OpenStack Docs: oslo.policy," 2017. https://docs.openstack.org/oslo.policy/latest/ (accessed Aug. 06, 2017).

[50] X. Jin, R. Krishnan, and R. Sandhu, "Role and Attribute Based Collaborative Administration of Intra-Tenant Cloud IaaS," in *Proceedings of the 10th IEEE International Conference on Collaborative Computing: Networking, Applications and Worksharing*, 2014, pp. 261-274, doi: 10.4108/icst.collaboratecom.2014.257591.

[51] R. Haemmerlé and Rémy, "On Combining Backward and Forward Chaining in Constraint Logic Programming," in *Proceedings of the 16th International Symposium on Principles and Practice of Declarative Programming - PPDP '14*, 2014, pp. 213-224, doi: 10.1145/2643135.2643144.

[52] J. Cheng, S. Nara, and Y. Goto, "FreeEnCal: A Forward Reasoning Engine with General-Purpose," in *Knowledge-Based Intelligent Information and Engineering Systems*, Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 444-452.

[53] OpenStack.org, "OpenStack Docs: Telemetry service overview," *OpenStack.org*, 2017. https://docs.openstack.org/mitaka/install-guide-rdo/common/get_started_telemetry.html (accessed Aug. 16, 2017).

[54] C. Ngo, Y. Demchenko, and C. de Laat, "Multi-tenant attribute-based access control for cloud infrastructure services," *J. Inf. Secur. Appl.*, vol. 27, pp. 65-84, Dec. 2015, doi: 10.1016/j.jisa.2015.11.005.

[55] D. J. Buehrer and C.-Y. Wang, "CA-ABAC: Class Algebra Attribute-Based Access Control," in *2012 IEEE/WIC/ACM International Conferences on Web Intelligence and Intelligent Agent Technology*, Dec. 2012, pp. 220-225, doi: 10.1109/WI-IAT.2012.268.

[56] N. Pustchi and R. Sandhu, "MT-ABAC: A Multi-Tenant Attribute-Based Access Control Model with Tenant Trust," in *International Conference on Network and System Security*, 2015, pp. 206--220, Accessed: May 24, 2016. [Online]. Available: http://link.springer.com/chapter/10.1007/978-3-319-25645-0_14.

[57] D. Lin, P. Rao, E. Bertino, N. Li, and J. Lobo, "Policy decomposition for collaborative access control," in *Proceedings of the 13th ACM symposium on Access control models and technologies - SACMAT '08*, Jun. 2008, p. 103, doi: 10.1145/1377836.1377853.

[58] P. Rao, D. Lin, E. Bertino, N. Li, and J. Lobo, "An algebra for fine-grained integration of XACML policies," in *Proceedings of the 14th ACM symposium on Access control models and technologies - SACMAT '09*, Jun. 2009, p. 63, doi: 10.1145/1542207.1542218.

[59] S. Menard and L. Nell, "JPype documentation — JPype 0.6.2 documentation," 2014. https://jpype.readthedocs.io/en/latest/ (accessed Aug. 05, 2017).

[60] Nullege, "jpype - Nullege Python Samples." http://nullege.com/codes/search/jpype (accessed Aug. 05, 2017).

[61] Y. A. Younis, K. Kifayat, and M. Merabti, "A novel evaluation criteria to cloud based access control models," in *2015 11th International Conference on Innovations in Information Technology (IIT)*, Nov. 2015, pp. 68-73, doi: 10.1109/INNOVATIONS.2015.7381517.

[62] V. Hu and K. Kent, *Guidelines for access control system evaluation metrics*. 2012.

[63] A. Corradi, M. Fanelli, and L. Foschini, "VM consolidation: A real case based on OpenStack Cloud," Futur. Gener. Comput. Syst., vol. 32, pp. 118-127, Mar. 2014, doi: 10.1016/j.future.2012.05.012.

[64] B. Tang and R. Sandhu, "Extending openstack access control with domain trust," *Netw. Syst. Secur.*, 2014, Accessed: Mar. 17, 2016. [Online]. Available: http://link.springer.com/chapter/10.1007/978-3-319-11698-3_5.

[65] D. MacKenzie, "Ubuntu Manpage: time - run programs and summarize system resource usage," *ubuntu.com*, 2010. http://manpages.ubuntu.com/manpages/xenial/man1/time.1.html (accessed Aug. 10, 2017).

[66] askubuntu.com, "command line - How can I measure the execution time of a terminal process? - Ask Ubuntu," *askubuntu.com*, 2011. https://askubuntu.com/questions/53444/how-can-i-measure-the-execution-time-of-a-terminal-process (accessed Aug. 10, 2017).

[67] J. Nielsen, *Usability engineering*. Academic Press, 1993.

[68] B. Taylor, A. K. Dey, D. Siewiorek, and A. Smailagic, "Using Crowd Sourcing to Measure the Effects of System Response Delays on User Engagement," in *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems - CHI '16*, 2016, pp. 4413-4422, doi: 10.1145/2858036.2858572.