International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395-0056
Volume: 10 Special Issue: | Apr 2023 www.irjet.net p-ISSN: 2395-0072

International Conference on Trends in Engineering & Technology- 2023 (ICRTET)
Organised by: VSM College of Engineering, Ramachandrapuram

Private Cloud Monitor

M.V.Ramana [1], T.D.Bhavani [2], D.Rajitha[3], M.Sumasri [4], T.Prasad [5], V.B.S.Ratna [6]

[1]Associate Professor, [2,-6]Students B.Tech. Computer Science Engineering,
V. S. M. College ofEngineering, Ramachandrapuram, A.P, India

Abstract - Authorization is an important security concern in cloud computing environments. It aims at regulating an access of the users to system resources. A large number of resources associated with Rest APIs typical in cloud make an implementation of security requirements challenging and error prone. To alleviate this problem, in this paper we propose an implementation of security cloud monitor. We rely on model- driven approach to represent the functional and security requirements. Models are then used to generate cloud monitors.The cloud monitors contain contracts used to automatically verify the implementation. We use Django web framework to implement cloud monitor and Open Stack to validate our implementation.

I. INTRODUCTION

In many companies, private clouds are considered to be an important element of data centre transformations. Private clouds are dedicated cloud environments created for the internal use by a single organization [20]. According to the Cloud Survey 2017 [3], private clouds are adopted by 72% of the cloud users, while the hybrid cloud adoption (both public and private) accounts for 67%. The companies, adopting private clouds, vary in size from 500 to more than 2000 employees. Therefore, designing and developing secure private cloud

Environments for such a large number of users constitute a major engineering challenge.

Usually, cloud computing services offer REST APIs (Representational State Transfer Application Programming Interface) to their consumers. REST APIs, e.g., AWS[1], Windows Azure , Open Stack , define software interfaces allowing for the use of their resources in various ways. The REST architectural style exposes each piece of information with a URI, which results in a large number of URIs that can access the system. Data breach and loss of critical data are among the top cloud security threats . The large number of URIs further complicates the task of the security experts, who should ensure that each URI, providing access to their system, is safeguarded to avoid data breaches or privilege escalation attacks.

Since the source code of the Open Source clouds is often developed in a collaborative manner, it is a subject of frequent updates. The updates might introduce or remove a variety of features and hence, violate the security properties of the previous releases. It makes it rather unfeasible to manually check correctness of the APIs access control implementation

II. PRIVATE CLOUDS WITH REST PRINCIPLES

Cloud computing promises to improve agility, achieve scalability, and shorten time to market [27] of software development. This vision relies on the use of REST APIs, which enable extensibility and scalability of any cloud framework. To facilitate extensibility, REST APIs expose their functionality as resources with unique Uniform Resource Identifiers (URIs). In complex systems, like the cloud frameworks, this results in a large number of URIs. Since the same set of HTTP methods (GET, PUT, POST, and DELETE) can be invoked on them, with different authorization rules,safeguarding such a large number of URIs is a challenging task. For example, let us consider a volume resource that is offered by the Cinder API of Open Stack [8]. Cinder is one of the services that is a part of the modular architecture of Open Stack. It provides storage resources (volume) to the end users, which can be consumed by the virtual servers [8]. A volume is a detachable block storage device that acts like a hard disk. Cinder API exposes the volume resource via (/{projected}/volumes/).

Any user of the project (e.g., project administrator, service architect or business analyst) with the right credentials can invoke the GET method on volume to learn its details. However, only the project administrator and service architect can update the existing volumes or add new volumes, and only the project administrator can delete a volume.

To offer scalability, REST advocates the stateless interaction between the components. This allows the REST services to cater to a large number of clients. Without storing the state between the requests, the server frees resources rather quickly that ensures system scalability. However, to construct the advanced scenarios using a stateless protocol, we should enforce a certain sequence of steps to be followed. Hence, we can treat such a behavior as a state full one, where the response to a method invocation depends on the state of the resource. For example, a POST request from the authorized user on the volumes resource would create a new volume resource if the project has not exceeded its share of the allowed volumes, otherwise it will not be created. Similarly, a DELETE request on the volume resource by an authorized user would delete the volume if it were not attached to any instance, otherwise it would be ignored.

III. CLOUD MONITORING FRAMEWORK

Figure 1 presents the overall architecture of the Cloud Monitoring Framework. A cloud developer uses IaaS to develop a private cloud for her/his organization that would be used

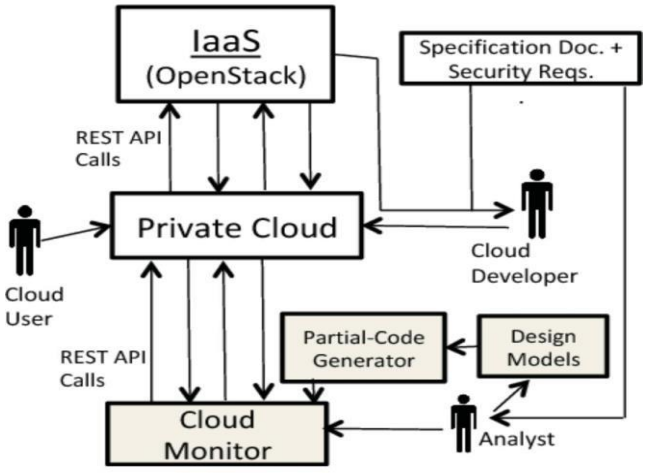

Fig. 1.Architecture of the Cloud Monitoring Framework

By different cloud users within the organization. In some cases, this private cloud may be implemented by a group of developers working collaboratively on different machines. The REST APIprovided by IaaS is used to develop the private cloud according to the specification document and required security policy.

The cloud monitor is implemented on top of the private cloud. The main original components of our work are highlighted as grey boxes in Figure 1. The security analyst develops the required design models based on the specification document and security policies. These models define the behavioural interface for the private cloud and specify its functional and security requirements.

A. Workflow

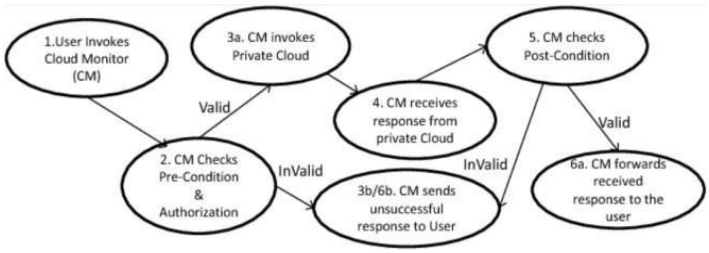

Fig. 2. Workflow in Cloud Monitor (CM)

Our cloud monitor acts as a proxy interface on top of the private cloud implementation. It interprets the response codes of different resources to analyse how the request went. HTTP has a list of status codes [15]. The HTTP response code is a numeric value that informs the clients whether the request has been processed successfully. For example, the value 200 meansthat the request was successful, 404 means the resource was notfound and 403 implies that it is forbidden to make this request on this resource.

B. Users of Cloud Monitor

The cloud monitor can be used in a variety of ways. Namely,the users of cloud monitor can be:

1) a cloud developer, who is implementing the cloud for his/her organization and interested in validating his/her implementation during the development phase with respect to functional and security requirements.

2) a tester, who is interested in testing whether the implementation of the cloud satisfies its design specifications and security requirements.

3) a security expert, who wants to validate whether the cloud implementation has any security loopholes that may give access rights to the unauthorized users or prevent the authorizedusers to access the resources.

4) an automated testing script, which uses CM as a test oracle and invokes the cloud implementation through the cloudmonitor to validate the authorization policy for all the resources. The invocation results can be logged for further faultlocalization.

In the next section, we present our design approach to specifying the behavioural interfaces for the RESTful architectures.

IV. DESIGN APPROACH

Our approach focuses on modeling APIs that are REST compliant [35]. We use UML (Unified Modeling Language) [38], which is well accepted both in industry and academia, and has many associated industrial-strength automated tools. We briefly describe the construction of our resource and behavioral models using Cinder component of Open Stack introduced in section II as an example.

Open Stack services define the permitted requests based on the access rules introduce in their policy. json files, which follow Role Based Access Control (RBAC) paradigm [17]. Similarly, to the other Open Stack services, Cinder uses Keystone service to validate the user's credentials and authorization requests [6].

A. Resource Model

We use the UML class diagram [38] with the additional design constraints to represent resources, their properties, and the relations between each other. We use the term resource definition to define a resource entity such that its instances are called resources. This is analogous to the relationship between a class and its objects in the object-oriented paradigm.

A collection resource definition is represented by a class with no attributes and a normal resource definition has one or more attributes. Each association has a name as well as minimum and maximum cardinalities showing the number of resources that can be part of the association.

B. Behavioural Model

The projects are created by the cloud administrator using Keystone and users or user groups are assigned the roles in these projects. It defines the access rights of the cloud users in the project. A volume can be created, if the project has not exceeded its quota of the permitted volumes and a user is authorized to create a volume in the project. Similarly, a volume can be deleted, if the user of the service is authorized to do so, and the volume is not attached to any instance, i.e., its status is not in-use.

V. CONTRACT GENERATION

The interface of a cloud service advertises the operations thatcan be invoked on it. A cloud developer finds the cloud service API on the web and integrates it with the other services by invoking the advertised operations and providing it with the required parameters.

These operations may imply a certain order of invocation orassume special conditions under which they can be invoked. Such conditions, i.e., pre- and post-conditions of a method, constitute contracts. This information together with the expected effect of an operation form a part of the behavioral interface of a service.

When the method *m* triggers a transition *t* in a state machine,the pre-condition for the method *m* should be true, i.e., the invariant of the source state of transition *t* and the guard on *to* evaluate to true.

For example, we are interested in generating a pre-condition to invoke the DELETE method on the *volume* resource, as shown in Figure 3 (right). DELETE on volume invokes three transitions in the behavioral model: one from the state project with volume and full quota and two from the state project with volume and not full quota. We should note that while there are three different transitions triggered by DELETE(volume), the actual implementation should combine the behavioral of these transitions into one method. Therefore, in order to generate themethod contract, we need to combine the information stated in all the transitions triggered by a method into a pre-condition andpost-condition for that method, as shown in the Listing 1.

Similarly, the post-condition states that if the pre-condition for invoking a method is true then its post-condition should alsobe true. We say that the post-condition of the method *m* is true if the conjunction of the state invariant of the target state of *t* and the effect on the transition *t* is true provided its pre- condition is true. Listing 1 shows the post-condition for DELETE(volume) method. The implication principle encompasses the stateful behavioral since the same method can be fired from different states of the system and have different results. Thus, if the method is fired with a certain