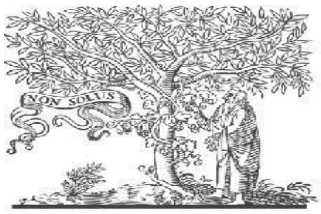
International Journal for Innovative Engineering and Management Research
A Peer Revieved Open Access International Journal
www.ijiemr.org

COPY RIGHT



ELSEVIER SSRN

IJIEMR Transactions, online available on 1st Jan 2021. Link
:http://www.ijiemr.org/downloads.php?vol=Volume-09&issue=ISSUE-12

DOI: 10.48047/IJIEMR/V09/I12/139

Title: GENERATING CLOUD MONITORS FROM MODELS TO SECURE CLOUDS
Volume 09, Issue 12, Pages: 818-822
Paper Authors

M.SIRISHA, M BHARATHI, MANCHALA RAMYA, K.SONY

International Journal for Innovative Engineering and Management Research
A Peer Revieved Open Access International Journal
www.ijiemr.org

GENERATING CLOUD MONITORS FROM MODELS TO SECURE CLOUDS

M.SIRISHA[1], M BHARATHI[2], MANCHALA RAMYA[3], K.SONY[4]

[1,2,3,] B TECH Students, Department of CSE, Princeton Institute of Engineering & Technology For Women, Hyderabad, Telangana, India.

[4] Assistant Professor, Department of CSE, Princeton Institute of Engineering & Technology For Women, Hyderabad, Telangana, 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 OpenStack to validate our implementation.

Keywords: UML diagram, dataset, scientific program, provenance.

1. INTRODUCTION

Open source cloud frameworks allow their customers to build their own private Infrastructure as a Service (IaaS). IaaS provides Virtual Machines (VMs) under the pay-per-use business model. The source code of Open Source (OS) clouds is distributed publicly. Moreover, often software is developed in a collaborative manner that makes it a subject of frequent updates. These updates might introduce or remove a variety of features and hence, violate the security properties of the previous releases. Assuring the security of opensource clouds is an important concern for cloud providers. Often open source clouds use REST architectural style to offer their APIs. REST offers a different architectural style to invoke remote services in contrast to contemporary SOAP-based services. Its different architectural style motivates the need to develop novel design and security assurance methodologies to handle its stateless protocol for developing stateful services. Stateful services can have different states that a service must go through during its lifecycle. It requires a certain sequence of method invocations that must be followed in order to fulfil the functionality a service promises to deliver to its users. In this work, we propose a methodology that consists of creating a (stateful) wrapper that emulates the usage scenarios and contains an explicit representation of security and functional requirements as contracts. We adopt a model-driven approach – Security and Rest compliant UML Models (SecReUM) – that builds on the theory presented in [22] to create a security-validating wrapper. We define the structural interface of a REST API using UML class diagram. The usage scenarios – the dynamic behaviours – are represented as state diagrams. These models lead to RESTful interfaces, describe the behaviour of operations in terms of

preconditions and post conditions and also facilitate the specification of the authentication mechanism. In this work, we demonstrate how to generate contracts defining the security properties as pre- and post-conditions using these models and implement them as a wrapper for the cloud implementation. The approach is implemented as a wrapper in Django Web Framework for the KeyStone component of OpenStack. OpenStack is an open-source software platform for cloud computing that offers REST interfaces to provide IaaS (Infrastructure as a Service) Keystone offers identity service in OpenStack for authentication and authorization.

2. LITERATURE SURVEY

Here let us consider a volume resource that is offered by the Cinder API of OpenStack [8]. Cinder is one of the services that is a part of the modular architecture of OpenStack. 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 (/{projectid}/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 stateful 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 is not attached to any instance, otherwise it would be ignored. The security requirements combined with the functional requirements specifying the conditions under which a method can be invoked and its expected output result in a large volume of information. Moreover, such information should be defined for each resource, which becomes overwhelming for any cloud developer. In addition, if an API is developed in a distributed manner, i.e., by several developers working on implementing different parts of API, then the design errors and inconsistencies become inevitable. Therefore, we should propose an automated approach that would facilitate implementing correct security policies for each resource of the system and assure that the right users have an access to the right resources.

3. RELATED STUDY

. A cloud developer uses IaaS to develop a private cloud for her/his organization that would be used 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 API provided by IaaS is used to

International Journal for Innovative Engineering and Management Research
A Peer Revieved Open Access International Journal
www.ijiemr.org

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 behavioral interface for the private cloud and specify its functional and security requirements. In addition, our design models define all the information required to build the stateful scenarios using REST as the underlying stateless architecture. In our approach, the construction of the design models serves several purposes:

1) The models specify the system from different viewpoints and hence, the security analysts can choose to specify in detail only those part of the system that they consider to be critical;

2) The models provide a graphical representation of the expected behavior of the system with the contracts, which can be communicated with a relative ease compared to the textual specifications;

3) The models serve as the specification document and facilitate reusability;

4) They are used to generate code skeletons with the integrated behavioral and security contracts; and finally,

5) We can use several existing model-based testing approaches to facilitate functional and security testing of private clouds. We build on our partial code-generation tool that is capable of generating the code skeletons from the design models. We extend this work by targeting the security requirements, i.e., the access rights over the resources, and propose an automated approach to representation the security requirements in the code. The generated code skeletons are then completed by the developer with the desired implementation of the methods.

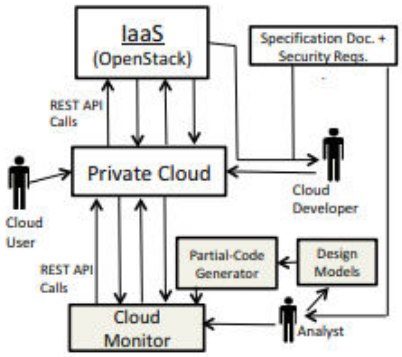


Fig.3.1. Architecture of the Cloud Monitoring Framework.

4. PROPOSED SYSTEM

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. We represent the behavioral interface of the REST API by a UML state-machine. Figure 3 (right) shows an excerpt from the behavioral interface of Cinder API for a project. It contains the information about the methods, which a user can invoke on the volume resource and the invocation conditions. In the example shown, at any given time a project can be only in one of three states. A project initially starts with no

International Journal for Innovative Engineering and Management Research
A Peer Revieved Open Access International Journal
www.ijiemr.org

volumes attached to it. A volume is added to the project by the POST request. The request method can only be triggered, if the user belongs to the user group admin or member. As a result, the project transits to the project with volume and not full quota state. The subsequent POST requests on the project will keep it either in the same state or transfer to the project with volume and full quota state, depending on the guard conditions. The DELETE method can only be invoked, if the status of the volume is not in-use and user belongs to the user group admin. The change of the project state depends on the guard conditions. We define the invariant of a state using OCL as a boolean expression over the addressable resources. In this way, the stateless nature of REST remains uncompromised because no hidden information about the state of the service gets stored between the method calls.

AUTHONTICATION:

Authorization in OpenStack, and other open source clouds is based on RBAC model. In RBAC, the access rights of a user are defined by his/her role. We assume that the information about the roles and the corresponding access rights to the resources is well-defined and available for the cloud developer and security analyst. In the current industrial practice, this information is usually given in a tabular format. We specify this information as the guards in the OCL format, which makes it amenable to an automated translation into the method contracts. In the behavioural model, each method should be labeled with a corresponding security requirement represented as a comment on a transition or state, as shown in Figure 3. When a state or transition with the requirement annotation is traversed, we get an indication which security requirement is met. This provides traceability of security requirements during the validation phase.

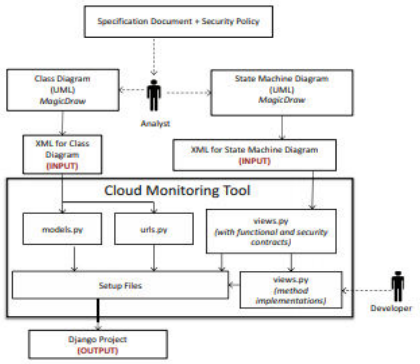


Fig.3.2. Cloud Monitor.

The current implementation continues our work on developing the wrapper. It focuses on validation of the authorization policy and its implementation in the cloud environment. The main steps in our implementation are as follows:

• We look for the resources in the class diagram to implement database tables in models.py. For each resource we create a table in the database, and analyze its associations to define their relationships with their keys. This creates a local copy of the resource structures as required by our monitor.

• urls.py contains the relative URLs of each resource and ways to access their respective views. This information is fully defined in the class diagram. By traversing the tags on the associations between the resources, we compose the paths of each resource. We always start from the corresponding collection, especially if we are referencing an item in the collection.

• The views.py file contains the main functionality of the system, i.e., the code that will run when accessing a resource through its URL according to the request (GET, PUT, POST or DELETE). These concepts are defined in the state machine diagram. The population of views.py is done in four steps:

1) Add information regarding the permitted methods over the resources;
2) Extract the functional contracts from the behavioral model as explained in section V and add them to the appropriate views;
3) Add the authorization information from the guards into the appropriate views;
4) Read security requirements from the comments on the transitions and add them as the corresponding variables in the code.

5. CONCLUSION

In this paper, we have presented an approach and associated tool for monitoring security in cloud. We have relied on the model-driven approach to design APIs that exhibit REST interface features. The cloud monitors, generated from the models, enable an automated contract-based verification of correctness of functional and security requirements, which are implemented by a private cloud infrastructure. The proposed semi-automated approach aimed at helping the cloud developers and security experts to identify the security loopholes in the implementation by relying on modelling rather than manual code inspection or testing. It helps to spot the errors that might be exploited in data breaches or privilege escalation attacks. Since open source cloud frameworks usually undergo frequent changes, the automated nature of our approach allows the developers to relatively easily check whether functional and security requirements have been preserved in new releases.

REFERENCES

[1] C. Perez, The Deep Learning A.I. Playbook : Strategy for Disruptive Artificial Intelligence,I.M.:Intution Machine, 2017

[2] V. Joaquin, S. Larisa, “Expos´e: An Ontology for Data Mining Experiments”, Third Generation Data Mining Workshop at ECML PKDD 2010.

[3] I. Mitsuru, S. Kazuhisa, K. Osamu, M. Riichiro, “Task ontology: Ontology for building conceptual problem solving models”, In proceeding of ECAI98 Workshop on Applications of Ontologies and Problem-Solving model, pp.126-133, ECA, 1998.

[4] A. F. Martins, R. A. F. De, “Models for Representing Task Ontologies”, Proceeding of the 3rd Workshops on Ontologies and their Application, 2008.

[5] S. Kanjana, S. Maleerat, “Ontology Knowledge-Based Framework for Machine Learning Concept”, iiWAS '16 Proceedings of the 18th International Conference on Information Integration and Webbased Applications and Services, pp. 50-53, 2016.

[6] P. Gustavo Correa, E. Diego, L. Agnieszka, P. Panče, S. Larisa, S. Tommaso, V. Joaquin, Z. Hamid, “ML Schema: Exposing the Semantics of Machine Learning with Schemas and Ontologies”, ICML 2018 - RML Workshop.

[7] G. Eason, B. Noble, and I. N. Sneddon, MEX Interfaces: Automating Machine Learning Metadata Generation, https://www.researchgate.net/publication/305143958, 2016.

[8] E. Diego, N. M. Pablo, M. Diego, C. D. Julio, Z. Amrapali, L. Jens, “MEX Vocabulary: A Lightweight Interchange Format for Machine Learning Experiments”, SEMANTiCS 2016 Proceedings of the 12th International Conference on Semantic Systems, pp. 17- 24, 2016.