Content Level Access Control for OpenStack Swift Storage

Prosunjit Biswas
Univ. of Texas at San Antonio
San Antonio, TX, USA
prosun.csedu@gmail.com

Farhan Patwa
Univ. of Texas at San Antonio
San Antonio, TX, USA
farhan.patwa@utsa.edu

Ravi Sandhu
Univ. of Texas at San Antonio
San Antonio, TX, USA
ravi.sandhu@utsa.edu

ABSTRACT

Swift, the object storage service from OpenStack cloud computing platform is used for storing, managing and retrieving large amounts of data. Inside Swift, uploaded files, also known as objects, are organized in containers. Objects inside a container are managed to be accessible or restricted from users through Access Control Lists (ACLs). Swift ACL, at the finest level, works on a Swift object enforcing who can or cannot access the object. Once an object is accessible to some one, he gets the full content of the object. Thus Swift ACL is an "all or nothing" approach.

In this work, we allow Swift users to specify access control at the content level of a Swift object. The content level policy describes who can access which part of a Swift object. When a request comes for downloading (i.e. read) an object, we check content level policy along with the ACL of the object. The response of the request is a partial content of the requested object based on the credential of the requester. Our prototype implementation is done on Swift objects of content type 'application/json'.

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Miscellaneous

Keywords

Content level access control; OpenStack Swift; ACL

1. INTRODUCTION

OpenStack Swift is a highly deployed open source cloud storage solution. With its unlimited storage capacity, it is used to store any number of large or small objects. In Swift terminology, uploaded documents are called objects. A user can upload or download an object using well defined APIs or available Swift client programs. But not everyone can download (i.e. read) every object stored in Swift. In order to maintain who can or cannot access an object, Swift uses Access Control Lists (ACLs). ACL specifies who can or cannot access an object. Unfortunately, the ACL based approach for Swift is an 'all or nothing' approach in a way that an user can either download (i.e. read) the whole object or cannot download it at all.

We propose a content level access control mechanism for objects stored in Swift. This approach lets Swift users specify who can access which part of a Swift object. To give a concrete example, consider that a hospital stores its patient records as Swift objects. These records should be accessed differently by different personnel. For example, 'doctors' can see certain part while the 'billing accountant' can see other part of the record. Our implementation would let the data publisher specify policies expressing who can see which part of the data.

Our prototype implementation is based on JSON formatted documents. We use JSON data because recently JSON has gained immense commercial popularity which is reflected by developments including JSON document database such as MongoDB supported by the OpenStack cloud platform, Twitter's latest API (v 1.1) which supports only JSON data and so on.

2. BACKGROUND

2.1 Swift

Swift is a highly available, distributed, eventually consistent object storage which can operate standalone or integrated with the rest of the OpenStack cloud computing platform. It is used to store lots of data efficiently, safely, and cheaply using a scalable redundant storage system [3]. As opposed to conventional storage architectures like file systems which manage data using file hierarchy and block storage, Swift manages data as objects. Each object typically includes the data itself, a variable amount of metadata, and a globally unique identifier.

Using its well defined RESTful API [5], users can upload or download objects to and from Swift storage. Inside Swift, objects are organized into containers which is similar to directories in a filesystem except that Swift containers cannot be nested. Again, a user is associated with a Swift account and can have multiple containers associated with the account. In order to manage user accounts, user containers and objects inside a container, Swift uses an Account Server, a Container Server and Object Servers correspondingly.

When a user corresponding to a user account requests for an object inside a container (either for uploading or downloading), the Account Server looks for the account first in its account database and finds associated containers with the

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
CODASPY'15, March 2–4, 2015, San Antonio, Texas, USA.
Copyright © 2015 ACM 978-1-4503-3191-3/15/03 ...$15.00.
http://dx.doi.org/10.1145/2699026.2699124.

account. The Container Server then checks the container database to find whether the requested object exists in the specified container and finally the Object Server looks into 'object databases' to find retrieval information about the object. In order to retrieve an object, the Proxy Server needs to know which of the Object Servers are storing the object, and path of the object in the local filesystem of that server.

2.2 Swift ACL

Once an object is stored in Swift, who can or cannot access the object is determined by Swift Access Control List (ACL). Swift has different levels of ACL—Account level ACL, and Container level ACL, for example. Container level ACL is associated with containers in term of a *read* action, or *write* action or *listing* action. If a user is authorized read action on a container through read ACL, he or she can read or download objects from the container. Similarly, write ACL enables uploading an object into a container and listing ACL enables the list operation on the container. Account ACLs, on the other hand, allow users to grant account-level access to other users. Of these two types of ACL, Container level ACL is finer grained in that different containers of a single account can be configured differently. Nonetheless, Swift ACL is limited in the following ways.

- Once an object is set accessible to someone, he or she gets the full content of the object. But there can be some sensitive information that the publisher wants to hide out.
- Swift ACL allows sharing an object with others, but it does not allow to share objects selectively at the content level.

2.3 JSON (JavaScript Object Notation)

JSON or JavaScript Object Notation is a data representation format which uses human readable text to represent data. In JSON, data is represented in one of two forms—as an *object* or as an *array of values*. A JSON object is defined as a collection of *key-value* or *attribute-value* pairs where a *key* or an *attribute*[1] is simply a string representing a name and a *value* which is either one of the following primitive types—string, number, boolean (true or false), null or another object or an array. On the other hand, an array is defined as a set of an ordered collection of *values* (as defined above) starting from index zero. The formal definition of JSON data format is given in [1]. JSON structure has following characteristics

1. A JSON document forms a hierarchical structure which is a rooted tree.
2. In the rooted tree, leaf nodes represent text data of the document and non-leaf nodes are used to give the data a name and thus organize it.
3. In the rooted tree, a node can be uniquely identified by traversing the document from the root node to the target node.

[1]To avoid confusion with the use of *attribute* for attribute-based access control we will exclusively use the term *key* in this paper.

```
{
  "personal_record":{
    "name":"Alice",
    "DOB": "1/1/1990",
    "identification":{
      "DL": "25526509",
      "SSN": "32433149"
    }
  },
  "employment_record":{
    "Designation": "employee",
    "salary":50000
  }
}
```

Figure 1: A sample JSON Document containing records of an employee.

3. LABEL BASED ACCESS CONTROL

In order to protect a JSON document stored as a Swift object, we assign each JSON item (i.e. value of a JSON key) an *object-label* and each user a *user-label*. Then we specify policies in the form of (*user-label* values, *action*, *object-label* values) which means that objects labeled with any of 'object-label' values are allowed to be accessed by the users labeled with any of 'user-label' values for the specific 'action'. Here we present an informal description of the model and its open source implementation is available in [4].

3.1 Model Components

In LaBAC (Figure 2), we have one attribute assigned to objects and one attribute assigned to users. Object attribute is named *object-label* and user attribute is named *user-label*. These attributes are set valued attributes and the values of the attributes may form a partial order.

Object: Object is any resource we want to protect with the model. Examples include a JSON document or items inside a JSON document.

Object-label: Object-label is the attribute assigned on the objects. The values of this attribute may form a partial order.

User and *user-label*: User-label is the attribute assigned on each user. In a simple case, user-label values can be the set of roles assigned to the user. The values of this attribute may form a partial order.

Action: Action is the list of available actions to be exercised on the objects.

Policy: A policy in this model is a tuple of (*user-label* values, *action*, *object-label* values). The policy is interpreted such that objects labeled with any of 'object-label' values are allowed to be accessed by the users labeled with any of 'user-label' values for the specific 'action'.

Attribute Hierarchy: In our model, both object-label and user-label values may form a hierarchy or more specifically a partial order. The effect of attribute hierarchy is shown in Figure 3. As we can see in the figure, if a policy allows an action for user-label l_{uj} on object-label l_{oj}, due to the attribute hierarchy, all users having a equivalent or senior label than l_{uj} can also access object-label l_{oj} or its junior labels.

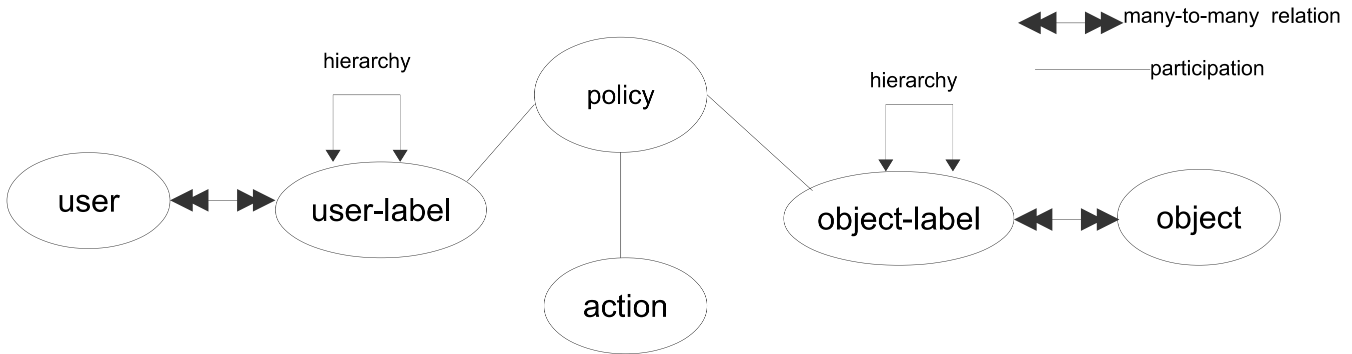

Figure 2: Label Based Access Control Model.

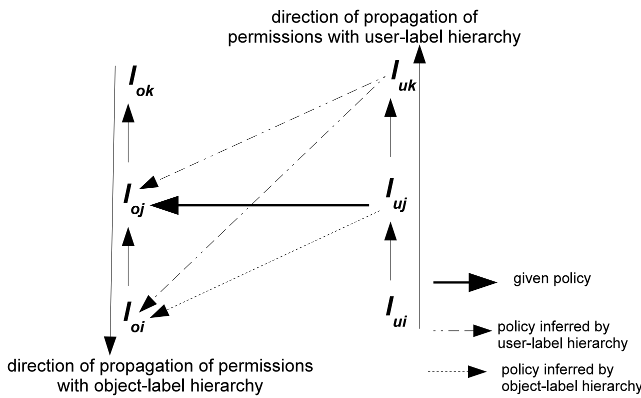

Figure 3: Propagation of permission with attribute hierarchy.

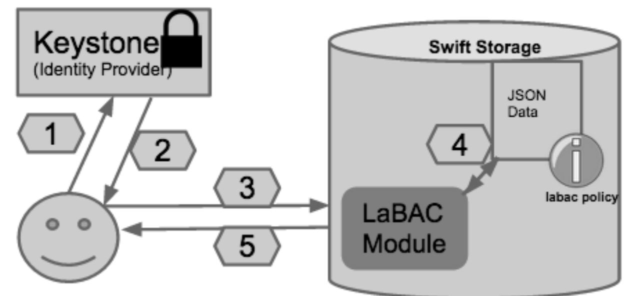

Figure 4: Required Changes in Swift Object Server for Our Extension

4. CONTENT LEVEL PROTECTION

Swift ACL specifies who can or cannot access a Swift object but it cannot specify who can access which part of the object. In order to specify security policies at the content level, Swift has to be aware of the content type and data format of the object. In our case, we addressed Swift objects of content-type *'application/json'* which is standard JSON format. How our protection mechanism works is summarized below.

- JSON items to be protected are identified using JSONPath. For example, SSN in JSON data given in Fig. 1 is identified using JSONPath '$.personal_record.identification.SSN'.
- *object-label* values are assigned on the specified JSON item. For example, to specify that SSN is a sensitive information, we assign a label *'sensitive'* on JSONPath '$.personal_record.identification.SSN'.
- We use LaBAC policy to specify who can access (read) which *object-label*. For example, if only users with user-label 'manager' can access *sensitive* information, then we specify the LaBAC policy *('manager', read, 'sensitive')*.

When using JSONPath to identify a JSON item, one may want to specify value at the path as a condition. For example, salary information (given in Fig. 1) is sensitive only if the salary is greater than 50,000. Furthermore, it is also possible to protect one item based on value of a different item. For example, identification information (specified by JSONPath '$.personal_record.identification') of a user is sensitive when his salary is greater than 50,000.

4.1 Labeling JSON Items

If the JSON document is large, labeling all JSON items can be tedious. In order to reduce labeling effort, we propagate label assigned on a JSON item to all its descendant items. For example, if the *personal_record* item (Fig. 1) is labeled *sensitive*, then all its descendant nodes (name, DOB, identification, DL, SSN) are also labeled *sensitive*.

5. IMPLEMENTATION

5.1 Changes in Swift Object Server

We have extended the logic of Swift Object Server. In the existing implementation, when a request comes for downloading of an object, Object Server checks the ACL and if the object is allowed by ACL, the Object Server reads the object from the disk and pass the whole content to the requester through Proxy-server.

In our implementation (see Figure 4), if the ACL denies the request, no further check is made and user gets corre-

sponding error messages. Otherwise, we retrieve the object, content level policy (stored with the Swift Object as metadata), user credential (user-label specifically which is the roles of the user maintained by Keystone) and pass them to the LaBAC module. LaBAC module processes the requested object based on the policy and user credential, and removes unauthorized content from the object. Then only the authorized partial content of the object is returned to the requester through Swift Proxy-server.

Note that in the implementation, we have used OpenStack Keystone [2] as the identity provider and we have mapped user roles provided by the Keystone as user-label values of the requester.

5.2 Storing of Policies

In our implementation, we have two different types of policies—LaBAC policies in the form of (*user-label* values, *action*, *object-label* values) and content-level policies in the form of *(JSONPath,* {*Labels*}*)*. All of these policies are stored as the metadata of the Swift object. Note that the Swift object is the JSON document itself.

One challenge of storing policies as metadata of Swift object is that Swift does not allow a single metadata item larger than 256 bytes. To circumvent this limitation, policies are stored as multiple metadata items.

5.3 Limitation of the Implementation

Our prototype implementation works only on objects of type 'application/json'. If requested object is not a JSON file or the requested object does not have content level policy set, the requester gets full content of the file.

6. PERFORMANCE

In order to analyze the performance of our implementation, we compared the download time of a Swift object enabling content policy and without enabling content policy. Our analysis (Figure 5) shows that our implementation works well for Swift object of size smaller than 100KB beyond which CLAC does not work efficiently. We believe this is because our implementation exhausts memory very soon. We conjecture that pre-labeling objects and enforcing access control in divide-and-concur fashion may improve performance.

7. RELATED WORK

There have been very few works for access control of JSON data, although JSON and XML data are very similar and lots of works has been done at the content level for XML data [7, 8]. Additionally there are prior works that apply object labels at the content level [6] for access control purposes. But, to the best of our knowledge, applying content level access control for the application context of OpenStack Swift has not yet been performed.

8. CONCLUSION

As more and more data is being uploaded in the cloud, data may contain sensitive information. With existing Swift API, one can either access the full content of an object or cannot access it at all. We propose an extension of Swift Object Server where someone can specify policies on a Swift object at the content level and let different users access different parts of it. We hope that this work would help Swift users to share objects effectively with others having more control at the content level.

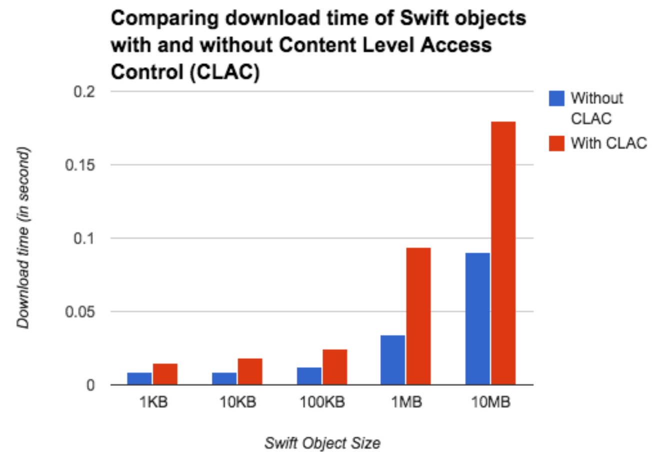

Figure 5: Performance of our Implementation

Acknowledgement

The authors acknowledge the support of Rackspace for this work.

9. REFERENCES

[1] JSON Official Website. http://json.org/. [Online; accessed 09/2014].

[2] Keystone, the openstack identity service. http://docs.openstack.org/developer/keystone/. [Online; accessed 09/2014].

[3] Openstack swift offical documentation. http://docs.openstack.org/developer/swift/. [Online; accessed 09/2014].

[4] Python Package - Label Based Access Control. https://pypi.python.org/pypi/labac/0.11. [Online; accessed 09/2014].

[5] Swift API Official Documentation. http://docs.openstack.org/api/openstack-object-storage/1.0/content/. [Online; accessed 09/2014].

[6] N. R. Adam, V. Atluri, E. Bertino, and E. Ferrari. A content-based authorization model for digital libraries. *Knowledge and Data Engineering, IEEE Transactions on*, 14(2):296–315, 2002.

[7] E. Bertino, S. Castano, E. Ferrari, and M. Mesiti. Specifying and enforcing access control policies for xml document sources. *World Wide Web*, 3(3):139–151, 2000.

[8] E. Damiani, S. De Capitani di Vimercati, S. Paraboschi, and P. Samarati. A fine-grained access control system for xml documents. *ACM Transactions on Information and System Security (TISSEC)*, 5(2):169–202, 2002.