AWS Identity-based Policies with "Read", "Write" and "Execute" Actions

by

Boyun Zhang

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Applied Science
in
Electrical and Computer Engineering

Waterloo, Ontario, Canada, 2020

Author's Declaration

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

This thesis addresses Amazon Web Service (AWS) identity-based policies with "read", "write" and "execute" actions. AWS is a large provider of cloud computing, security is an important property that an application running in AWS must meet. Towards this, AWS provides users with their services, a powerful mechanism and associated syntax, to articulate identity-based policies which manages and grants permission to an identity includes the IAM user, group or role. The current design for AWS policy syntax requires the specification, by the owner of cloud application, of the actions that users or role can be allowed to execute. While file system with traditional UNIS permissions also manages resources in a manner similarly to AWS but with three actions only: "read", "write" and "execute". We propose a new syntax for AWS identity-based policy that all the possible actions are restricted to "read", "write" and "execute". We expect this new syntax will be more usable than the current design from the standpoint of ease and accuracy. We discuss the design and carry out a small-scale human participant study with 20 participants to validate this hypothesis. The result of study demonstrates that current specifying AWS policy helps AWS community developers easier to adhere least-privilege and brings users more convenience on access control.

Acknowledgements

I would like to thank my supervisor, Mahesh Tripunitara, for his continuing engagement and help on my master thesis. I would also like to express my thank to the committee members,Professor Werner Dietl and Professor Derek Rayside, for taking their time to review my work and provide with professional suggestions. Last but not least, I would like to express my sincere gratitude to my parents for all their support and encouragement in the way of pursuing my master's degree.

Dedication

I want to give a special dedication to my parents, a sincere gratitude to my father and mother who have always support me during this process.I hope that this accomplishment will make you all proud.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Within decades, increasing number of developers are using cloud provider to develop and run applications as a Software-as-a-Service (SaaS) to make benefit of cloud computing. Cloud computing offers a scalable and flexible network access to a shared pool of a storage computing resources. The resource can be provisioned and the user is allowed to purchase the resource they are using.[22] Business are seeking different ways to reduce the hardware and management costs, while the cloud providers are improved its adoption according to this. With the popularity of the using of cloud service, one of an inevitable problem based on this the security problem for different cloud users.

Given an security policy helps to better prepare for the security challenges. It's a specification of how to implement security principles and technologies. Amazon Web Services (AWS) has become the largest cloud computing provider in the past few years. And in order to let developers make control of every execution, the AWS Identity and Access Management(IAM)[9] plays an important role during the process. The service provides both authentication and authorization. IAM has a notion of policy which is a high level representation of the actions a user is allowed to perform on resources. It's a web service helps securely control access to actions and resources for each services. The permissions granted by a policy rely on the interactions of different statements and conditions. The policy can either grant access or revoke access to the specific resources. In addition, the statement conditions can be passed on access details such as the MFA (multi-factor authentication) and other configuration options.

1.1 Problem Statement

AWS provides constructs which is a high level representation of the actions IAM user, group or role. The policy provide the permission for an identity to control what actions the identity can perform on the specific resources with conditions.

The right permissions is the first priority for users to grant in their policy to finish the needed actions. Due to the fact that the users may not fully take into account the semantics of the policy language, even some users have used the heuristic-based syntactic checks that detect policy patterns: the use of a wildcard but the syntactic checks ate unsound. While others are attempting to enumerate all the possible actions to a policy but find this intractable and complex.

While according to our observation, the miscellaneous amounts of actions of each AWS services will make trouble and cause some security troubles like over-privilege for developers when controlling access to the resource.

1.2 Our Work

In this study, we got the inspiration from the traditional UNIX permission of the file system, which only consider "read", "write" and "execute" actions to control each action access to the resource. And developing a new format of the AWS identity-based policy output to let AWS community developers make an easier access control method. And we also investigate the usability of the newly proposed AWS identity-based policy format, which is how easy for an AWS developer use the identity-based policy to make access control to the needed actions on the specific resources with conditions.

Matthew mentioned that "Previous research on the use of least privilege practices in the context of operating systems revealed that the overwhelming majority of study participants did not utilize least privilege policies."[5] This was due to their partial understanding of the security risks, as well as a lack of motivation to create and enforce such policies. Failing to create least privilege policies in a cloud computing environment is especially high risk due to the potentially severe security consequences. Which explain an usual issue happened in our cloud services nowadays, finding a way to reduce the happening of over-privilege in the AWS and other cloud service's access control policy is becoming pressing. The modification should not be only on the AWS customers but also on the AWS policy themselves.

Gaoshou explained that "Permission/privilege division is the essential way to solve systematic security problem. In another word, security performance of a system is dependent

on the way of permission/privilege division"[24]. This suggests that the permission division plays an important role of solving the policy security problem. While the popularity of traditional UNIX permissions with the "read", "write" and "execute" permission combinations will greatly decrease the complexity of the current AWS IAM policy permissions.

Based on the commonly-happened over-privilege problem in the cloud service access control policy and the advantage of the concise file system permission, we have point out a design hypothesis:

Combining the file system with traditional UNIX permission with the AWS identity-based policy will help developer decrease the occurrence possibility of the AWS access-control security problem.

We also designed an usability study which participants are trained separately on the AWS currently-used identity-based policy and our modified AWS identity-based policy with "read", "write" and "execute" actions. Then we showed the example of the identity-based policy generation based on the given code snippets. Finally several code snippets were given and asked the participants to generate the identity-based policy based on their previous training and understanding.

1.3 Outline

The chapters of the thesis are organized as below.

In Chapter 2 we focused on different file system operations which include but not limit to create, modify, delete, rename, get content from file/ directory. We have construct a file system example with traditional UNIX permission. By simulating different actions of file system with traditional UNIX permission from C language and enumerating each possible permission combination from each layer of the structure, we have figured out the least available permission to achieve the execution.

In Chapter 3, we made a brief introduction of AWS IAM, another introduction on the structure of general IAM identity-based policy will be provided which include statements, actions and resources(ARN). Then provided the method for mapping the AWS service resource to the file system according to the file system hierarchy with traditional UNIX permissions. Which include the way of determine the root directory and the distinguish of each layer. Based on the previous research on the permission of file system actions, we analyzed the achieved goals and the using resources for each action from AWS services, and instead with "read", "write" and "execute" actions, we also simplified the output of the identity-based policy, which only contain "read", "write" and "execute" in the

Action element. Finally, we discussed the expected compiler generated policy based on the resource, and due to the fact that some actions need more permissions than the others, the compiler outputs all the possible actions with the given resource and the permission for each layer of the directory. We used the AWS service Elastic Transcoder for the case study for the compiler.

In Chapter 4, we performed a small-scale human participant study which used to record the accuracy and time for task completion, each individual task was aimed to generate the AWS identity-based policy based on the giving AWS service. The participants were randomly divided into two group to have different training materials, one group was trained on the AWS general structure of the identity-based policy with actions and resources, while the other was given an introduction of the mapping procedure on AWS service resource to the file system with traditional UNIX permission and the AWS identity-based policy with "read", "write" and "execute" actions. Each group was given a policy generation example by the given code snippet. After training, the participant completed several tasks based on the action on single service or several actions on different services. The detailed information will be provided by section which include: The qualification of the participants, the setup for the study, training procedure , the introduction of each task and corresponding solution. We had also made the analysis based on the small-scale human participant study result.

Chapter 2

Study of File System and Permission Control

In this chapter, the study includes the introduction of different file systems with traditional UNIX permission and Access Control List(ACL) permissions, the permission control with traditional UNIX permission and Andrews File System(AFS), and the mechanism of basic operations in the file system. Then a file system is created as an example, and procedures are explained to figure out the least privilege for the operations.

2.1 File System and Permissions

In this chapter, a concise overview of the different file systems with traditional UNIX permissions and Access Control List(ACL) permission are presented. Which include the file system structure,components and permission control.

2.1.1 File Systems with traditional UNIX permissions

Unlike ACL-based systems, permissions on Unix-like systems are not inherited. Files created within a directory do not necessarily have the same permissions as that directory.

A file system is the combination of files and directories in a hierarchy. File system may be individually added or deleted from the file and directories in global *namespace*. A specific location is allocated for each file system in *namespace*. File system has a hierarchical file structure as it contains a root directory and its sub-directories. All other directories can

be accessed from the root directory. A partition usually has only one file system. The root directory of file system can reach to this location.The parent directory of all file systems is located in the root of *namespace* "/", which is called the root file system. Root file system is always existed in file system. The end of each file system branch contains the leaves or the regular file.

File systems are usually existed physically in the hard disks which always reside on the block storage devices. On the other hand, Linux support per-process *namespace*, which allow each process to have the access to the file and directory optionally. Each process is inherited the *namespace* of its parent by default, but the process is also allowed to create its own *namespace* which will take the owner of several storage location and a specific root directory.

There are different kinds of file systems with traditional UNIX permission such as Ext, Ext2, Ext3 and Ext4. In this thesis, we are focused on the file system with traditional UNIX permissions.

2.1.2 Components

File is the most basic and fundamental abstraction in the file system. "Everything is file" is the basic theory followed by the file system with traditional UNIX permissions. Apart from file, the directory, hard link and special file are the other commonly used components in the file systems.

2.1.2.1 Regular Files

The most common opponent in file system, which contain bytes of data and organized into a linear array called the byte stream. There is no solid format for file system to define as a regular file. And also not enforced any structure in file except tree byte stream in system level[19].

Regular file is usually accessed through filename, but they are not associated with name. A regular file usually referenced by an inode, which is a data structure that describes the attributes and the file disk block locations of the file. Usually the inode stores the metadata which we called the attributes of the file include the file permission, file timestamp, size and the other file attributes. The kernel can access to the regular file through the inode number.

2.1.2.2 Directory

A File system cataloging structure which contains references to other computer files or other directories. The regular file can opened from user space by a name not a inode number. Directory is acted as a mapping of readable names to the inode numbers which the directory contained to viewed like the normal file.

There is only one directory "/" on the disk initially. In a hierarchical file system, a directory contained inside another directory is called a sub-directory.[3] The terms "parent" and "child"" are often used to describe the relationship between a sub-directory and the directory in which it is cataloged, the latter being the parent. The top-most directory in such a file system, which does not have a parent of its own, is called the root directory[19].

Although the directories are acted as normal files, the kernel only allowed them to be manipulated by using some special system calls. These system calls allow for the adding and removing of links, which are the only two sensible operations anyhow.

2.1.2.3 Hard Links

Hard links are directory entries that associate names with files in the file system. All the directory-based file systems should have at least one hard link giving the original name for each file. Hard links also allow multi path names point to the same data in the file system. Hard link can either be in the same directory or in two more directories.

2.1.2.4 Special Files

Special file is a type of file store in the file system, also called as device file. For the file system with traditional UNIX permissions, there are four types of special file: character device file, named pipes, Unix domain sockets and block device files. Special files are a way to let certain abstractions fit into the file system. Special files usually provide simple interfaces to standard devices, but can also be used to access specific unique resources on those devices, such as disk partitions.

2.1.3 File System Management and Traditional Unix Permission

For the file systems with traditional UNIX permissions, multiple users are allowed to work on the same server without disrupting each other. While individuals share the file access will cause the risk or the loss of data if other users secretly access to the files or

directories. File system have added the permission to specify the user privilege which is given to each file or directory.

2.1.3.1 File/Directory Ownership

There are three types of owner for each file or directory in the file system with traditional UNIX permissions.

- **User**: The user who creates the file, which also becomes the owner of the file.
- **Group**: The group is created to have the same permission to file or directory, all users in the group will be given the same access to file or directory.
- **Other** The permission of all the other user to file or directory. The user have neither create the file nor belong to any of the group.

2.1.3.2 Traditional UNIX Permission

Every file and directory in the file system with traditional UNIX permissions have the combinations of the following three permissions:

- **Read**: The permission given the user to open and read the content of the file. Read permission on the directory given the user ability to list all the filenames belong to the directory.
- **Write**: The permission given the user ability to modify the contents of the file. Write permission on the directory authorize the user to rename, delete or create the file from the directory.
- **Execute**: The permission allows the user to execute and run the file. The execute permission on a directory enable the affected user to enter the directory, and access the belonging files and directories.

2.1.3.3 Default File/Directory Permission

Umask is a command that determines the newly created file/directory permissions. It's also a function that sets the user mask. The user mask is a grouping of bits formed as four-digit octal number, each number restricts the permission on the corresponding group, the format for the user mask is as below:

- The first digit sets permissions for the user.
- The second sets permissions for group.
- The third sets permissions for other, also referred to as "world".

The pre-defined initial permissions for files and directories are 666 and 777 respectively. The default umask permissions for root user and remaining users are 022 and 002 respectively. Which means that without any change, all files created by user root will get 644 permissions as default, the file will have read and write permission on the user group, but only read permission on the other two groups. For files, the default granted permission are 755, which means read and execute permission for everyone and also write permission for the owner of the directory.

2.1.3.4 Permission Checking and Modification

After get used to the ownership and the permission, user have another ability to check and change the access permission to the file or the directory[25].

User can can use the command **'ls-l'** to check all the files or directories permission under the current path.

User can use command **'chmod'** to change the file or directory permission on user, group or other.

Below an example is given to describe the process of the checking and changing the directory permission.

1. We first use the command **'ls-l'** to check the all the file/directory permission under the current path, and find that we have a directory **'user1'** which have the permission: **'read-write-execute'** on user.

Figure 2.1: Permission before the modification

2. Then, use the **'chmod'** command to change the *'user1'* permission, and only grant the **'execute'** permission to user.

```
~boyun$ chmod u=x user1
```

Figure 2.2: Permission modification

3. Finally we check again the for the file/directory permission under the current path, and find that the permission of directory *'user1'* have changed successfully.

```
~boyun$ ls -l
total 8
d--xr-xr-x 2 peterzhangby peterzhangby 4096 Sep 17 15:09 user1
```

Figure 2.3: Permission after the modification

2.1.4 Types of File Systems

There are different types of file systems with traditional UNIX permissions. File systems can be classified into disk file system, network file system(NFS) and file systems from other Unix systems.

2.1.4.1 Disk File System

Disk file systems are file systems which manage data on permanent storage devices. File system usually store physically. For this type of file system, it take advantage of the ability of disk storage media to randomly address data in a short amount of time while also permits multiple users (or processes) access to various data on the disk without regard to the sequential location of the data. The examples include: ext2, ext3, ext4, etc.

2.1.4.2 Network File System

NFS is regarded as a distributed file system protocol allowing user on the client computer to access files over the computer network. Programs using local interfaces can transparently create, manage and access hierarchical directories and files in remote network-connected computers.

2.1.5 Andrew File System

AFS is a distributed file system designed to be similar to the UNIX file system, with security and scalability as the major goal. It enables users to share remote file in an easier way that AFS users can share all the files under the **/afs** root directory after given appropriate privileges. Kerberos is used for authentication, and implements access control list(ACL)s on directories for users and groups.

2.1.5.1 Permission

There are seven standard AFS ACL permission,and they can be divided into directory and file based on the function.

The four directory permissions are meaningful with the respect to the directory itself.

- **Lookup(l)**: Allows a user to list the files names and sub-directories in the AFS directory, obtain complete status information for the directory element itself.
- **Insert(i)**: Enables the user to add new files or sub-directories to the directory.
- **Delete(d)**: This enables the user to remove files and sub-directories from the directory or move them into other directories.
- **Administer(a)**: This permission enables a user to change the directory's ACL.

The three permissions below are more meaningful on the files or sub-directories in a directory.

- **Read(r)**: Grants users to get file content of files in a directory and list files in sub-directories.
- **Write(w)**: Enables users to modify files or change file content in a directory.
- **Lock(k)**: This permission lets users to run programs that issue system calls to lock files in the directory.

AFS provides eight additional permissions (A-H) that do not have a defined meaning, the permissions can be assigned by the system administrator based on the written application programs.

2.2 Least Privilege for the Operations in File System

In this section, we have designed a file system with traditional UNIX permissions, the verification procedures for the least privilege of the actions are introduced.

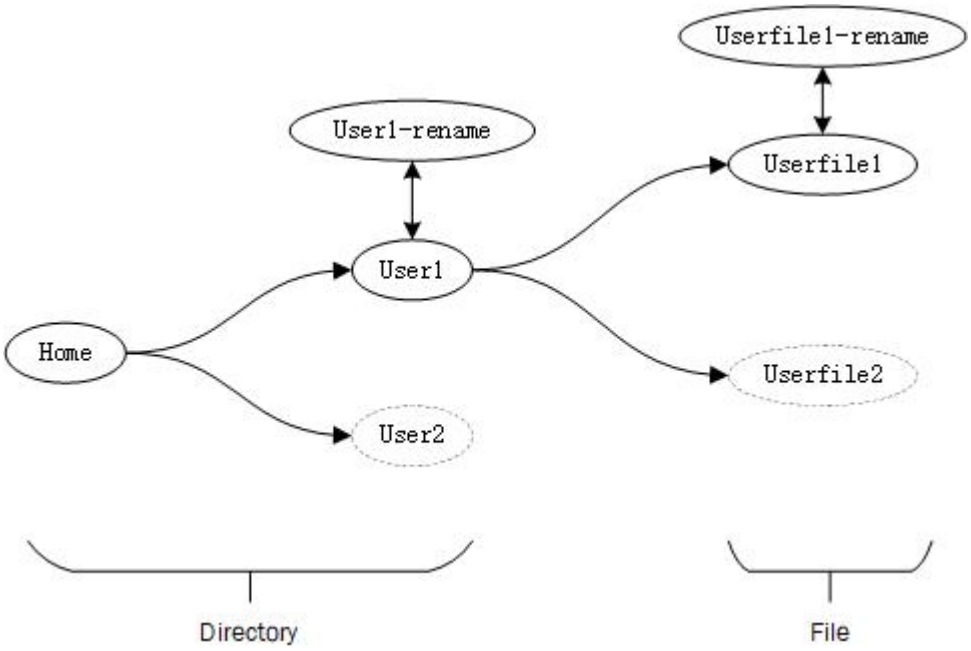


Figure 2.4: File System Example

At the top level of the system, we use *'Home'* at the root directory. Which initially contained a sub-directory *'User1'*. Inside the sub-directory which also contained the initial file *'Userfile1'*.

We have set the actions below to figure out the least privilege for the operations appeared in the file system with traditional UNIX permissions.

- **Create** Function on Directory: Create another sub-directory *'User2'*.
- **Create** Function on File: Create another file *'Userfile2'* under sub-directory *'User1'*.
- **Delete** Function on Directory: Delete sub-directory *'User2'*.
- **Delete** Function on File: Delete file *'Userfile2'*.

- **List** Function on Directory: Get file names under sub-directory *'User1'*.
- **Read** Function on File: Get file content on file *'UserFile1'*.
- **Rename** Function on Directory: Rename sub-directory *'User1'* to *'User1-rename'*.
- **Rename** Function on File: Rename file *'Userfile1'* to *'Userfile1-rename'*.
- **Edit** Function on File: Edit the file content in file *'Userfile1'*.

We have created a C script with system calls from C library to realize the operations above. We enumerated the file/directory permission on each layer in file system and figure out the least privilege to achieve each operations. A table have generated to illustrate the least privilege of each layer for the operation.

Operation	Home	Home/User1	Home/User2	Home/User1/Userfile1	Home/User1/Userfile2
Create-Directory (User2)	(w+x)	(-)	(-)	(-)	(-)
Create-File (Userfile2)	(x)	(w+x)	(-)	(-)	(-)
Delete-Directory (User2)	(w+x)	(-)	(-)	(-)	(-)
Delete-File (Userfile2)	(x)	(w+x)	(-)	(-)	(-)
List-Directory (User1)	(x)	(r)	(-)	(-)	(-)
Read-File (Userfile1)	(x)	(x)	(-)	(r)	(-)
Rename-Directory (User1-rename)	(w+x)	(-)	(-)	(-)	(-)
Rename-File (Userfile1-rename)	(x)	(w+x)	(-)	(-)	(-)
Edit-File (Userfile1)	(x)	(x)	(-)	(w)	(-)

Table 2.1: Least Privileges of the File System Operations

For the table above, we can conclude that for *create*, *delete* and *rename* action, the permission is only needed to grant on parent directory for the target file and directory. Each of the above three actions is needed the same least privilege for completing the operation. While for *edit* and *read* action, the permission is not only be granted on all the parent directories but also the target file and directories.

Thus through the example, we get the understanding of the least permission in order to control the file system actions, which have built a foundation for our thesis research on mapping the AWS service resource to file system and figure out the least privilege to AWS service actions.

2.3 Conclusion

In this chapter, we get a concise overview of file system with traditional UNIX permission, We also create a file system template with traditional UNIX permission and figure

out the least possible permission to achieve the actions for the file system with traditional UNIX permissions. In next chapter, we will start the study of the access control on AWS, and propose an optimized output for the AWS identity-based policy based on the file system permission research of this chapter.

Chapter 3

AWS Identity-based Policy Design

In this chapter, we analyze the identity-based policy on AWS and propose a different syntax. Firstly, an introduction of the AWS IAM service is given, which include the control management of the service, resource control and identity-based policy structure. Later we are focus on the resource of AWS service Elastic Transcoder and introduce the mapping procedure to the file system with traditional UNIX permissions. Then we propose a modification for the AWS identity-based policy syntax. A compiler is given to generated the modified AWS identity-based policy with "read", "write" and "execute" actions for the given AWS service resources and actions.

3.1 AWS IAM

Amazon web service(AWS) have provided the service IAM(identity and access management) helping user securely control access to the service resource. The service provide not only authentication but also authorization. The IAM is a Role-based access control management(RBAC)[7]. In IAM, user can implement RBAC by creating different polices and attached to different identity which include IAM users, groups of users, or IAM roles.

AWS IAM have also provided another authentication strategy called Attribute-based access control(ABAC) which is no longer needed for user to update and modify the current polices when the access resources changed.[6] The permissions for the new resources are automatically granted based on the attributes.

The access management portion of AWS IAM which is also regarded as authorization helping user to define what actions are allowed for the role in the AWS services. User can

also manage access in AWS IAM by creating and attaching policies to the IAM identities which include the role, users, the groups of the users or AWS services resources. When the policy is associated with the resource or an identity, the permissions can be defined. AWS makes an evaluation of the created policies when a principle requested by the identities. The users of IAM are regarded as identity in the service.[23] When a new IAM user is created, there is no permission granted until user create and attach the policy to the IAM user. IAM user can also organized to the IAM groups, when attaching a policy to the IAM group, all the members in group will be granted the same permissions by the policy. Users and groups can attach multiple polices which will grant different permissions. The permission of the user or the groups are depended on the combination of all the granted policies.

For managing the AWS resources, user usually needs to grant different AWS services to finish the task. For example, user needs to use an AWS event-driven, serverless computing platform given by AWS(Lambda Function) to accomplish the image processing which stored in the AWS key-value and document database service(DynamoDB)[16]. For accomplishing the actions above, the user needs to grant Lambda functions permissions to edit the DynamoDB service. While combining with AWS IAM policy, the process can be simplified. User can create the policy to the IAM role, and the role grants the Lambda Function permissions accessing to the DynamoDB. By using the IAM policy and IAM role, user do not need to embed credentials in code and can tightly access control to the expected functions in the service. For Figure 3.1,the IAM access policy is authorized to grant the required permissions to the DynamoDB table and CloudWatch Logs(an AWS monitoring and observability service)[13]. The policy is attached to the role, and this role will then be attached to a Lambda function, which will assume the required access to DynamoDB and CloudWatch Logs.

The architecture in figure 3.1 is using the Lambda Function to made read API calls such as 'GET' and write API calls such as 'UPDATE' to the DynamoDB table[15], and make write calls to CloudWatch in order to create the log files and record the log process during the period. All the service calls above are granted and attached by the IAM created policy.

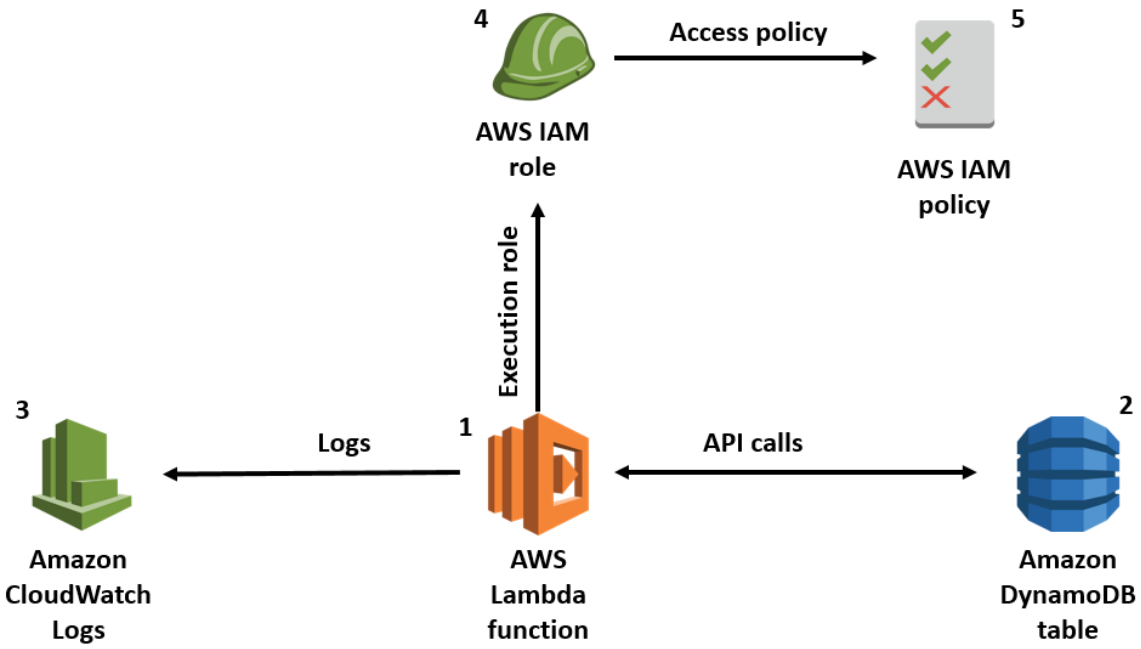


Figure 3.1: An IAM example to create the AWS IAM policy to decide whether AWS Lambda function have permission accessing to the AWS DynamoDB table, regard the serverless application Lambda function as a role, the IAM create the policy and attached to Lambda to grant permission to the DynamoDB service, the policy also allow Lambda function write functions to the AWS CloudWatch Logs.

For the example in Figure 3.1, the architecture of the solution using read API calls such as 'GET' and write API calls 'UPDATE' to modify the DynamoDB table. The Lambda function also create the log file and record the log process during the period to CloudWatch. All the service calls above are granted by created and attached IAM policies[15].

On the other hand, the resource request to the service is also authenticated by IAM as well. [21]IAM policy are divided into two types based on the different authentication purpose to the Identity-based policies which is used to control what actions the identity can perform, on which resources, and under what conditions, the Resource-based policies control what actions a specified principal can perform on the resource and under what conditions. For the purpose of this thesis, we are focus on the identity-based policy.

3.1.1 Identity-based Policy

AWS Identity-based policy managed the granted permission to an identity which include the IAM user, group or role. The identity-based policy can be further divided into inline policies and the managed policy. For the inline policy, the policy are created and attached for the specific user, group or role. While other identities do not have the ability to attach it[10].

A managed policy can be further classified as AWS managed policy and customer managed policy. The AWS managed policy is created and managed by AWS, user have no other permission but only attach the policy to different identities. The customer managed policy can be fully created and controlled by the user, which not only include modify the policy but also attach the policy to different user, role or groups. When the modification occurred to the policy, all the identity which the policy previously attached is taking the effect.

Most identity-based policies are stored in AWS as JSON documents. While the user can be assisted by the AWS IAM visual editor to create and edit policy.

For the JSON structure of identity-based policy, it's consists of three mandatory components and four other optional components. The three mandatory components are as below:

- **Effect**: Using **Allow** or **Deny** to indicate whether is policy is allowed or not to the including actions and resources.
- **Action**: The list of service actions which is allowed or denied affected by *Effect*.
- **Resource**: The service resource which the actions apply. Format as ARN (Amazon Resource Names).

The rest optional components are as below:

- **Version**: The policy language version, the commonly used version is the latest 2012-10-17 version.
- **Sid**: The specific ID to differentiate statement.
- **Statement**: Use as the container to the policy components of *Effect*, *Action*, *Resource* and *Condition*.
- **Condition**: Specify the circumstances and qualifications for the granted policy.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "S3-CreateBucket",
            "Effect": "Allow",
            "Action": "s3:CreateBucket",
            "Resource": "arn:aws:s3:::bucket-test"
        }
    ]
}
```

The identity-based above allow an action '*CreateBucket*' from AWS service Simple Service Storage(S3) on the specific AWS S3 bucket '*bucket-test*'. While the identity-based policy can be expanded with various statements which include lists of actions from different services. The policy permission is the union for all the allowed or denied statements.

3.1.2 ARN

Amazon Resource Paths is an unique way to represent the resource path in AWS IAM identity-based policy. ARN is needed to specify resource precisely across AWS.

The general format of ARN is represented as below:

arn:partition:service:region:account-id:resource

Some resources omits the '*region*', '*account-id*' or both. The function of each components of ARN is as below:

- **partition**: Represent the location of the resource, each AWS account scoped to one partition.
- **service**: The namespace of the AWS service identification.
- **region**: The belonging area for the service, some service may not be available in specific regions according to the official user guide.
- **account-id**: The specific digits ID which represent the AWS account that own the resources.

- **resource**: The identifier of the resource, this part can be the name or the path which partitioned by slashes of the resource.

ARN path can include the wildcard (*) to match all the available resource in the partition. While for the ARN component '*principle*', the wildcard is not supported to represent all the users.

3.2 Resource Mapping

In this section, we are using the AWS resource types table and combining the theory of file system in Section 2.1.1 to map the AWS service resource to file system with traditional UNIX permissions. The process of the mapping procedure is provided. In this section, we are using AWS service Elastic Transcoder[8] for the case study.

Elastic Transcoder is a media operating service provided by AWS which allows user to format and convert media file from AWS S3[8]. User can create a job for the work of encoding under the pipeline which is used to manage the transcoding jobs. The preset provide the template that contain most of the setting for transcoding media file.

The resource type table list all the resource types that user can specify as the ARN in '*Resource*' element for AWS identity-based policy. The resource type can also define condition keys which the policy can be included.

Resource types	ARN
job	arn:${Partition}:elastictranscoder:${Region}:${Account}:job/${JobId}
pipeline	arn:${Partition}:elastictranscoder:${Region}:${Account}:pipeline/${PipelineId}
preset	arn:${Partition}:elastictranscoder:${Region}:${Account}:preset/${PresetId}

Figure 3.2: Elastic Transcoder Resource Types

Figure 3.2 represents the resource types of Elastic Transcoder, according to the definition of ARN and ARN format, all the three resource types keep identical except the partition after the last component which represent the resource path for Elastic Transcoder ARN, we regard the whole part before the last component '*Resource*' as the root directory in the file system. Each layer of the subdirectory and the regular file for file system can be

separated based on slashes (/) for Elastic Transcoder resource path, the leaves of the path is automatically became the regular file according to the definition of the file system.[20]

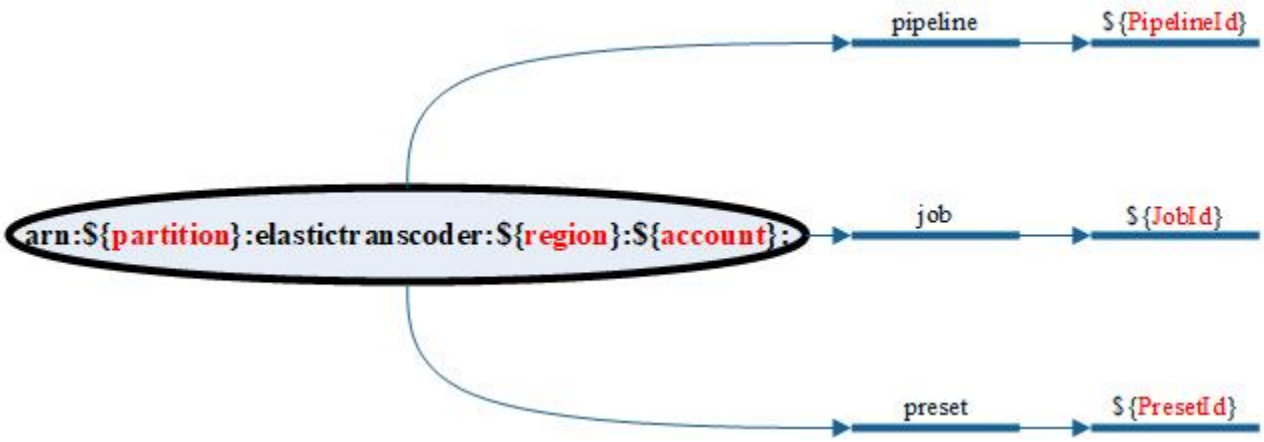


Figure 3.3: Mapping Elastic Transcoder Resource Type to the file system with traditional UNIX permissions

The portions in Figure 3.3 which preceded by the label ($) should be replaced by the actual values or the policy variable depending on the user setting. Each service may have different mapping of the file system depending on the constitution of resource type and the format of service ARN.

3.3 Identity-based Policy Output Optimization

In this section, we are combining the research of file system with traditional UNIX permissions and the Mapping of AWS resource type in order to make the optimization on AWS identity-based policy[9]. We are keep using the AWS service Elastic Transcoder as the case study and focus on the actions of Elastic Transcoder to figure out the corresponding operations on the file systems with traditional UNIX permissions.

Based on the permission control in file system with traditional UNIX permissions and the syntax of the AWS identity-based policy, we have made the hypothesis as below:

- Based on the access control of traditional UNIX permissions, we set the '*Effect*' element as '**Allow**'.

- The possible set of component '*Action*' are restricted to "read", "write" and "execute" which follow the access control rule in the file system with traditional UNIX permissions.
- The component '*Resource*' and the component '*Conditions*' is keep following the original policy syntax.

According to the hypothesis given above, we have gone through the actions in Elastic Transcoder, for each identity-based policy generated by the IAM visual editor with the current syntax, we have figured out the operation with same semantics on how the file system manage the resource with actions "read", "write" and "execute".

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "Elastic-Read-Preset",
            "Effect": "Allow",
            "Actions": [
                "elastictranscoder:ReadPreset"
            ],
            "Resources":[
                "arn:aws:elastictranscoder:region:
                account:preset/apple-mp4"
            ]
        }
    ]
}
```

The policy above is an identity-based policy '**Elastic-Read-Preset**' with the current design of syntax, which use the Elastic Transcoder action '**ReadPreset**' to get the preset configuration content on preset named '**apple-mp4**'.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Actions": [
                "read"
            ],
            "Resources":[
                "arn:aws:elastictranscoder:
                region:account:preset/apple-mp4"
            ]
        },
        {
            "Effect": "Allow",
            "Actions": [
                "execute"
            ],
            "Resources":[
                "arn:aws:elastictranscoder:
                region:account:preset"
            ]
        }
    ]
}
```

Followed the hypothesis we propose above, the least privilege on the operations of the file systems with traditional UNIX permissions in Chapter 2 and the mapping of the Elastic Transcoder Resource Type to the file system with traditional UNIX permissions, we can conclude that the '**ReadPreset**' action in Elastic Transcoder has the same semantic of the **'Read-File'** Function in the file system, we have generated the policy based on the syntax we suppose above with least privilege on the same service resource ARN.

3.4 Compiler

Based on the study of the AWS identity-based policy optimization, the compiler is conducted in order to bring convenience to the AWS community developer and help better understanding of the policy optimization. The design is based on transferring the optimized policy to the policy in currently-used AWS syntax. The compiler input is based on three mandatory components of the optimized policy which include '*Action*' and '*Resource*'. Due to the fact that the component '*Effect*' is set as **Allow** as default based on the hypothesis for the optimized policy, the compiler no longer consider it as the input element. The compiler expected output contain two mandatory components '*Action*' and '*Resource*' based on the AWS currently-used policy syntax, the component '*Effect*' is keep setting as **Allow** in current design[2].

According to our experiment, the scope of the resources and the actions containing "read", "write" and "execute" of some AWS service actions is wider than the other, the compiler will generate all possible sets of actions based on the input in order to help developers reducing the possibilities when accessing control to the actions and resources.

We are keep using the system calls from the service Elastic Transcoder as the case study, make a summary for all the actions of Elastic Transcoder in AWS, and according to their corresponding resource types we divide the actions to three types: **Preset**, **Pipeline** and **Job**.

For each resource type, the input resource is strictly followed ARN syntax, while the actions is authorized to "read", "write" and "execute" in each portion of the resource path on Elastic Transcoder. For each input with the same action combinations, the resource scope of the input is compared the wider range including wildcard with more restrictive range which contain the specific resource ID.

		Input				Output	
						Action Set	**Resource**
1	a	**Resource**	arn:aws:elastictranscoder:*:*:	preset/	*	DeletePreset	arn:aws:elastictranscoder:*:*:preset/*
		Action		(w+x)	(-)		
	b	**Resource**	arn:aws:elastictranscoder:region:account:	preset/	presetId	DeletePreset	arn:aws:elastictranscoder:region:account:preset/presetId
		Action		(w+x)	(-)		
2	a	**Resource**	arn:aws:elastictranscoder:*:*:	preset/	*	CreatePreset, DeletePreset	arn:aws:elastictranscoder:*:*:preset/*
		Action		(w+x)	(w)		
	b	**Resource**	arn:aws:elastictranscoder:region:account:	preset/	presetId	CreatePreset, DeletePreset	arn:aws:elastictranscoder:region:account:preset/presetId
		Action		(w+x)	(w)		
3	a	**Resource**	arn:aws:elastictranscoder:*:*:	preset/	*	ReadPreset	arn:aws:elastictranscoder:*:*:preset/*
		Action		(x)	(r)		
	b	**Resource**	arn:aws:elastictranscoder:region:account:	preset/	presetId	ReadPreset	arn:aws:elastictranscoder:region:account:preset/presetId
		Action		(x)	(r)		
4	a	**Resource**	arn:aws:elastictranscoder:*:*:	preset/	*	ListPresets, ReadPreset	arn:aws:elastictranscoder:*:*:preset/*
		Action		(r+x)	(r)		
	b	**Resource**	arn:aws:elastictranscoder:region:account:	preset/	*	ListPresets, ReadPreset	arn:aws:elastictranscoder:region:account:preset/*
		Action		(r+x)	(r)		

Table 3.1: Compiler Input and Output on the Elastic Transcoder's Resource Type: Preset

		Input				Output	
						Action Set	**Resource**
1	a	**Resource**	arn:aws:elastictranscoder:*:*:	Pipeline/	*	ReadPipeline	arn:aws:elastictranscoder: *:*:Pipeline/*
		Action		(x)	(r)		
	b	**Resource**	arn:aws:elastictranscoder: region:account:	Pipeline/	PipelineId	ReadPipeline	arn:aws:elastictranscoder:region :account:Pipeline/PipelineId
		Action		(x)	(r)		
2	a	**Resource**	arn:aws:elastictranscoder:*:*:	Pipeline/	*	ListPipelines, ReadPipeline	arn:aws:elastictranscoder: *:*:Pipeline/*
		Action		(r+x)	(r)		
	b	**Resource**	arn:aws:elastictranscoder: region:account:	Pipeline/	*	ListPipelines, ReadPipeline	arn:aws:elastictranscoder:region :account:Pipeline/*
		Action		(r+x)	(r)		
3	a	**Resource**	arn:aws:elastictranscoder:*:*:	Pipeline/	*	DeletePipeline	arn:aws:elastictranscoder: *:*:Pipeline/*
		Action		(w+x)	(-)		
	b	**Resource**	arn:aws:elastictranscoder: region:account:	Pipeline/	PipelineId	DeletePipeline	arn:aws:elastictranscoder:region :account:Pipeline/PipelineId
		Action		(w+x)	(-)		
4	a	**Resource**	arn:aws:elastictranscoder:*:*:	Pipeline/	*	UpdatePipeline, UpdatePipeline Notifications, UpdatePipeline Status	arn:aws:elastictranscoder: *:*:Pipeline/*
		Action		(x)	(w)		
	b	**Resource**	arn:aws:elastictranscoder: region:account:	Pipeline/	PipelineId	UpdatePipeline, UpdatePipeline Notifications, UpdatePipeline Status	arn:aws:elastictranscoder:region :account:Pipeline/PipelineId
		Action		(x)	(w)		
5	a	**Resource**	arn:aws:elastictranscoder:*:*:	Pipeline/	*	CreatePipeline, DeletePipeline, UpdatePipeline, UpdatePipeline Notifications, UpdatePipeline Status	arn:aws:elastictranscoder: *:*:Pipeline/*
		Action		(w+x)	(w)		
	b	**Resource**	arn:aws:elastictranscoder: region:account:	Pipeline/	PipelineId	CreatePipeline, DeletePipeline, UpdatePipeline, UpdatePipeline Notifications, UpdatePipeline Status	arn:aws:elastictranscoder:region :account:Pipeline/PipelineId
		Action		(w+x)	(w)		

Table 3.2: Compiler Input and Output on the Elastic Transcoder's Resource Type: Pipeline

		Input				Output	
						Action Set	**Resource**
1	a	**Resource**	arn:aws:elastictranscoder:*:*:	job/	*	CancelJob	arn:aws:elastictranscoder: *:*:job/*
		Action		(w+x)	(-)		
	b	**Resource**	arn:aws:elastictranscoder: region:account:	job/	jobId	CancelJob	arn:aws:elastictranscoder: region:account:job/jobId
		Action		(w+x)	(-)		
2	a	**Resource**	arn:aws:elastictranscoder:*:*:	job/	*	ReadJob	arn:aws:elastictranscoder: *:*:job/*
		Action		(x)	(r)		
	b	**Resource**	arn:aws:elastictranscoder: region:account:	job/	jobId	ReadJob	arn:aws:elastictranscoder: region:account:job/jobId
		Action		(x)	(r)		
3	a	**Resource**	arn:aws:elastictranscoder:*:*:	job/	*	CreateJob, CancelJob, ListPipelines, ReadPipeline, ListPresets, ReadPreset	arn:aws:elastictranscoder: *:*:job/*
		Action		(w+x)	(w+x)		
		Resource	arn:aws:elastictranscoder:*:*:	preset/	*		arn:aws:elastictranscoder: *:*:preset/*
		Action		(r+x)	(r)		
		Resource	arn:aws:elastictranscoder:*:*:	pipeline/	*		arn:aws:elastictranscoder: *:*:pipeline/*
		Action		(r+x)	(r)		
	b	**Resource**	arn:aws:elastictranscoder: region:account:	job/	jobId	CreateJob, CancelJob, ListPipelines, ReadPipeline, ListPresets, ReadPreset	arn:aws:elastictranscoder:region :account:job/jobId
		Action		(w+x)	(w+x)		
		Resource	arn:aws:elastictranscoder: region:account:	preset/	presetId		arn:aws:elastictranscoder:region :account:preset/presetId
		Action		(r+x)	(r)		
		Resource	arn:aws:elastictranscoder: region:account:	pipeline/	pipelineId		arn:aws:elastictranscoder:region :account:pipeline/pipelineId
		Action		(r+x)	(r)		

Table 3.3: Compiler Input and Output on the Elastic Transcoder's Resource Type: Job

In the case study above for Elastic Transcoder, the actions and resource types we considered is restricted to the policy attached to the application, while for more complex uses of attached policies, such as granting access to services, user is able to call the AssumeRole API [12]. This API call returns a set of temporary credentials that the service can use in subsequent API calls. Actions attempted with the temporary credentials have only the permissions granted by the associated role.

Chapter 4

Small-scale Human Participant Study

A small-scale human participant study was conducted for 22 participants. The aim of the study was for the usability of AWS identity-based policies with "read", "write" and "execute" actions. The study had received ethics clearance from the Office of Research Ethics prior to the study start.

The design of the study is followed by a between-participant mode. While the first two participants were regarded as pilot participants. 10 of the participants took part in the study with the currently-used AWS identity-based policy structure and syntax, while the other 10 participants used the modified AWS identity-based policy with "read", "write" and "execute" actions. Each group of the participants were given training materials and finished three tasks for generating AWS identity-based policy with the provided code snippets on one or more AWS services.

4.1 Participants

The study participants were the upper-year undergraduates and graduate students from computer engineering and computer science at the University of Waterloo. The invitations were sent via email and on the community bulletin board. In order to make the test result as persuasive as possible, a participant should be familiar or have experience on the basic operations of file system with traditional UNIX permissions and are conversant with UNIX permissions in the file system. The participants should possess no prior knowledge of identity-based policies in Amazon Web Services (AWS), nor know anything about serverless applications in AWS. We also anticipated the participants have the ability to quickly

learn the AWS identity-based policy or the basic operation of Linux command line. Each participant got $20 for the reward.

4.2 Study Setup

The study was conducted on a Microsoft Teams Live Video Chat. Only the participant and the study conductor were presented for the whole process of the study in the live chat. Due to the aim of the data collection, before the start of the study, the participant signed a consent letter which indicate the agreement of the audio collection, screen sharing during the whole process of the study and share the solution through the online editable word document which were used for recording their thinking process after the study. All the actions above were only for the use of study and would not send to anybody for other use. Each participant was given a hard copy of the training materials and the tasks. During the task, the participant was only allowed using the web page and resource as below:

- The actions of the service which is using in the task.
- Resources for the needed service.

After the study, we recorded the final policy the user created, and gave a confidence rating as that indicated how confident the participant was in having completed the task accurately.

4.3 Training

Before the commencement for the task, both sets of the participants had trained on their corresponding study objective. Users were initially shown the following:

- A brief introduction of the background and the aim of the study.
- Instructions on how to think aloud

For participants got trained on the currently used AWS identity-based policy, the training tasks were as follows:

- Participants were given an introduction of the AWS identity-based policy, which include the structure and the resource path.

- An example was given for generating an AWS identity-based policy according to the provided code snippet.

For another part of the participants, the training tasks were involved the following:

- A brief review of the basic operation on file system with traditional UNIX permissions based on the Command line and the permission control in the file system supported by Linux Operating System.
- An introduction of the mapping procedure from the AWS service resource to file system with traditional UNIX permissions.
- An introduction of the modified AWS identity-based policy syntax with "read", "write" and "execute" actions.
- An example was given for generating an AWS identity-based policy according to the provided code snippet.

All the training procedures were recorded as training videos, and provided watching to the participants during the study to guarantee the fairness. Participants were allowed to ask questions during the training, the questions were the clarifications about the technical aspect of the study.

4.4 Tasks

The participants were given three tasks to complete. The tasks were based on the *aws-serverless-shopping-cart* application, which was a serverless application simulate the shopping cart with the functions such as list cart, delete from cart, get cart total amount. The application had a total of ten lambda functions and the code snippet for each task was selected from these functions. For each task, a code snippet was given and the participant was asked to generate an AWS identity-based policy based on the code snippet. The difficulty of the task was increased by order and the subconscious bias were not accounted for the experiment.

An appendix was provided to participants for the task solving, the appendix included all the AWS services which were appeared in code snippets. Each service contained two tables. The resource table can be used to the mapping of AWS service resource to the file system, while the action table included all the needed actions from the code snippets with descriptions and the corresponding resource types.

The task description and the possible solutions for each group are provided below. Depending on the syntax and the structure of the AWS identity-based policy, the solution are not the only possible one, we accept any solution which meet the objection for the code snippet.

The mapping for all the appeared AWS service in code snippets to file system with traditional UNIX permissions are as below:

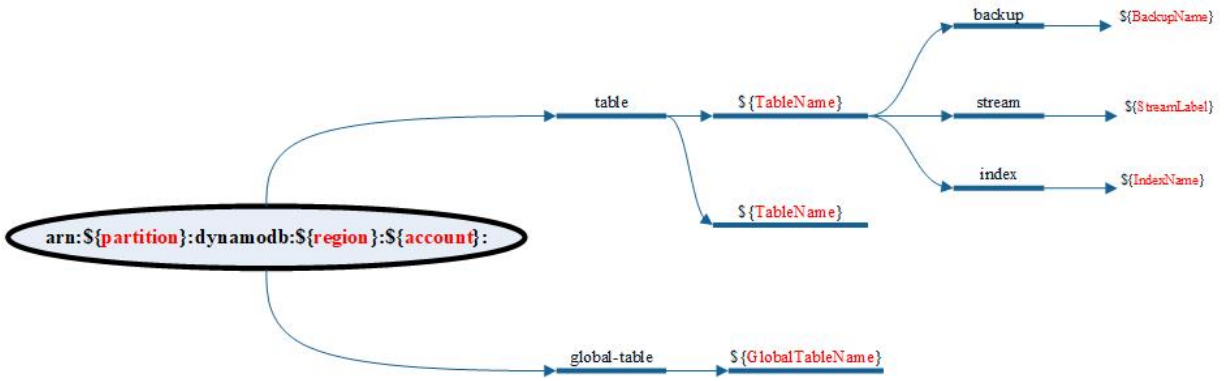


Figure 4.1: Mapping DynamoDb Resource Type to the file system

Figure 4.2: Mapping SQS Resource Type to the file system

The portions of figure which preceded by the label ($) should be replaced by the actual values or the policy variable depending on the user setting.

4.4.1 Task 1: Get Cart Item

The code snippet for Task 1:

```
dynamodb = resource('dynamodb')
def lambda_handler(dynamodb):
    response = dynamodb.GetItem(
        TableName='item-name'
        )
    return response
```

Solution for the ***currently used AWS identity-based policy***: Having an action GetItem from the AWS service: **DynamoDb**, using the Dynamodb resource type: Table.

```
{
    "Statement": [{
        "Effect": "Allow",
        "Actions": [
            "dynamodb:GetItem"
        ],
        "Resources":[
            "arn:aws:dynamodb:region:accountId:table/item-name"
        ]
    }]
}
```

Solution for the ***modified AWS identity-based policy with "read", "write" and "execute" actions***: According to the mapping of the AWS service resource to file systems with traditional UNIX permissions and the action descriptions, GetItem need *"read"* action for the resource type : Table on the target table and *"execute"* from all its parent directory.

```
{
    "Statement": [{
        "Effect": "Allow",
        "Actions": [
            "execute"
        ],
```

```
            "Resources":[
                "arn:aws:dynamodb:region:accountId:table"
            ]
        },
        {
            "Effect": "Allow",
            "Actions": [
                "read"
            ],
            "Resources":[
                "arn:aws:dynamodb:region:accountId:table/item-name"
            ]
        }]
    }
```

4.4.2 Task 2: Checkout Cart

The code snippet for Task 2:

```
dynamodb = resource('dynamodb')
def lambda_handler(dynamodb):
    response = dynamodb.CreateTable(
        TableName='item-category'
        )

    item_in_cart = response.get('Items')
    for item in item_in_cart:
        dynamodb.DeleteItem(
                TableName='item-category',
                        )
    return response
```

Solution for the ***currently used AWS identity-based policy***: Having two actions CreateTable and DeleteItem from the AWS service: **DynamoDb**, using the Dynamodb resource type: Table.

```
{
    "Statement": [{
```

```
            "Effect": "Allow",
            "Actions": [
                "dynamodb:CreateTable",
                "dynamodb:DeleteItem"
            ],
            "Resources":[
                "arn:aws:dynamodb:region:accountId:table/item-category"
            ]
        }]
}
```

Solution for the ***modified AWS identity-based policy with "read", "write" and "execute" actions***: According to the mapping of the AWS service resource to file system with traditional UNIX permissions and the action description, CreateTable need the *"write"* action for the resource type : Table on its parent directory, while the action DeleteItem need *"write"* action on the target table, *"execute"* from all the parent directories.

```
{
    "Statement": [{
        "Effect": "Allow",
        "Actions": [
            "execute"
        ],
        "Resources":[
            "arn:aws:dynamodb:region:accountId:table"
        ]
    },
    {
        "Effect": "Allow",
        "Actions": [
            "write"
        ],
        "Resources":[
            "arn:aws:dynamodb:region:accountId:table", "arn:aws:dynamodb:re
        ]
    }]
}
```

4.4.3 Task 3: Migrate Cart

The code snippet for Task 3:

```
dynamodb = resource("dynamodb")
sqs = resource("sqs")

def update_table(dynamodb):
    dynamodb.UpdateTable(
        TableName="item-category"
    )

def lambda_handler(dynamodb,sqs):
    sqs.UpdateQueue(
        QueueName="update-success"
        )
    response = dynamodb.DeleteItem(
        TableName="item-category"
        ItemName="item1"
        )
    return response
```

Solution for the ***currently used AWS identity-based policy***: Having two actions UpdateTable and DeleteItem from the AWS service: **DynamoDb**, using the Dynamodb resource type: Table. Also have an action DeleteQueue from the AWS service: **SQS**, using the SQS resource type: Queue.

```
{
    "Statement": [{
        "Effect": "Allow",
        "Actions": [
            "dynamodb:UpdateTable",
            "dynamodb:DeleteItem"
        ],
        "Resources":[
            "arn:aws:dynamodb:region:accountId:table/item-category"
        ]
    },
```

```
        {
            "Effect": "Allow",
            "Actions": [
                "sqs:UpdateQueue"
            ],
            "Resources":[
                "arn:aws:sqs:region:accountId:update-success"
            ]
        }]
}
```

Solution for the ***modified AWS identity-based policy with "read", "write" and "execute" actions***: According to the mapping of the AWS service resource to file system with traditional UNIX permissions and the action description, UpdateTable and DeleteItem need *"write"* action for the resource type : Table on the target table, *"execute"* from all the parent directories. For the AWS service **SQS** action UpdateQueue, *"write"* action is needed for the resource type : Queue on the target queue.

```
{
    "Statement": [{
        "Effect": "Allow",
        "Actions": [
            "execute"
        ],
        "Resources":"arn:aws:dynamodb:region:accountId:table"
    },
    {
        "Actions": [
            "write"
        ],
        "Resources":[
            "arn:aws:dynamodb:region:accountId:
            table/item-category"
            "arn:aws:sqs:region:accountId:
            update-success"
        ]
    }]
}
```

4.4.4 Procedure

Each group of participants were given the same task code snippet for the task, with an extra appendix containing all the actions and resources for the services used in the task for the reference. Each task statement was presented in the Microsoft Word file and was shared through Microsoft OneDrive. After the participant finished each task, the complement time was calculated. The order of the tasks was same for all participants. The participants were asked to record the confidence for the answer. After finished all tasks, the participants were asked about their experience of the whole process for the study.

4.4.5 Ethics Clearance

Due to the fact that this study involved human participants, the study has been reviewed and received ethics clearance through University of Waterloo Research Ethics Committee. The whole session for the study was recorded for analysis and further development for the study area. But all the collected data was kept anonymous for the privacy protection.

Subject: Study in usable computer security; $20 for 1 hour

Hello,

(This message is being sent on behalf of the researchers.)

We are conducting a study in usable security, for which we seek human participants. Each participant will be remunerated $20 for 1 hour of their time. A participant needs to meet the following prerequisites:

1. *Basic knowledge of UNIX/Linux read-write-execute file permissions.*
2. *Basic knowledge of programming in some programming language.*
3. *No prior knowledge of identity-based policies in Amazon Web Services (AWS).*
4. *No prior knowledge of serverless applications in AWS.*

You will be provided training for which you will need the ability to download and play a short MP4 movie. You will then be asked to complete a set of tasks for which you will need a text editor. The session will be conducted entirely remotely, via MS Teams. We will ask that you share your screen, and we will record the session. Your anonymity will be preserved as set out by the Office of Research Ethics here at the University of Waterloo. This study has been reviewed and received ethics clearance through a University of Waterloo Research Ethics Committee.

Please contact me, Boyun Zhang, boyun.zhang@uwaterloo.ca, for more information or to schedule a session. I am an MASc student in Electrical and Computer Engineering (ECE) under the supervision of Prof. Mahesh Tripunitara who can be reached via email to tripunit@uwaterloo.ca.

Sincerely,
Boyun Zhang

Figure 4.3: Recruitment Email for the Study

Information Letter for Usability of AWS identity based policies
with "read," "write" and "execute" actions

You are invited to participate in a research study conducted by Boyun Zhang, under the supervision of Prof. Mahesh Tripunitara of the University of Waterloo, Canada. The objectives of the research study are to assess the usability of AWS identity-based policies with "read," "write" and "execute" actions. The study is likely to be part of an MASc thesis.

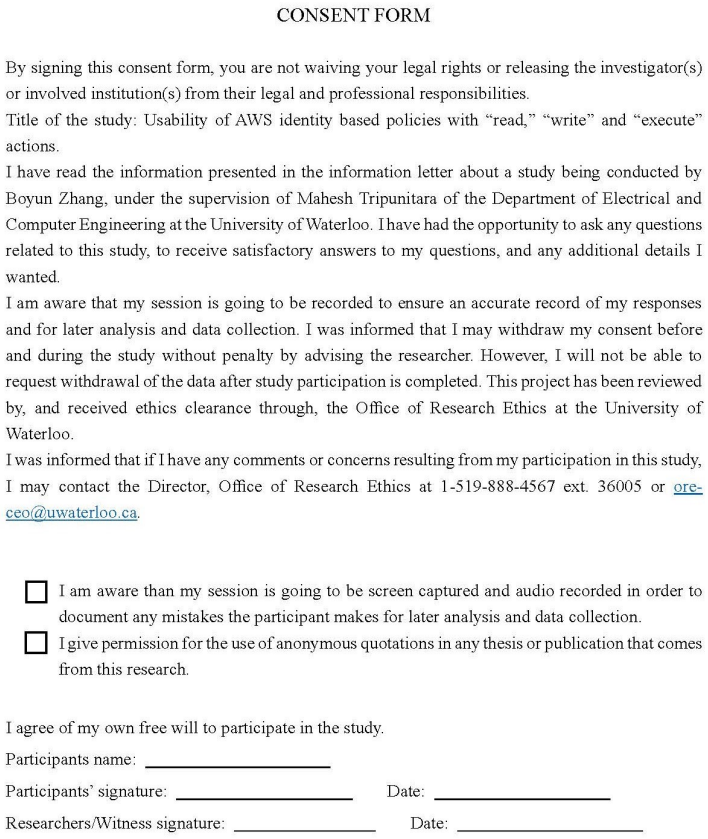
If you decide to volunteer, you will be asked to complete a 1 hour session remotely via Microsoft Teams. 20 minutes will be spent on training, and the remainder on some tasks we ask you to perform. With your permission, we will record the session for later analysis and collection of data, such as how long you took to complete a particular task, whether you appeared to have difficulty and for us to ensure that we accurately record the results.

We require participants to have basic knowledge of UNIX/Linux read-write-execute file permissions, basic knowledge of programming in some programming language, no prior knowledge of identity-based policies in Amazon Web Services (AWS) and no prior knowledge of serverless applications in AWS.

Participation in this study is voluntary. You may decline to answer any questions that you do not wish to answer, and you can withdraw your participation at any time. There are no known or anticipated risks from participating in this study. Any information about you will be kept confidential. All of the data will be aggregated and no individual will be identifiable from the aggregated results. The data, with no personal identifiers, collected from this study will be maintained on a password protected computer database in a restricted access area of the University. As well, the data will be electronically archived after completion of the study and maintained for two years and then erased. Should you have any questions about the study, please contact either Boyun Zhang (boyun.zhang@uwaterloo.ca) or Mahesh Tripunitara (tripunit@uwaterloo.ca). Further, if you would like to receive a copy of the results of this study, please contact either investigator.

I would like to assure you that this study has been reviewed and received ethics clearance through the Office of Research Ethics at the University of Waterloo, ORE#42626. However, the final decision about participation is yours. If you have any comments or concerns resulting from your participation in this study, please contact the Office of Research Ethics, at 1-519-888-4567 ext. 36005 or ore-ceo@uwaterloo.ca.

Figure 4.4: Introduction and Instructions for the Study in Consent Letter



CONSENT FORM

By signing this consent form, you are not waiving your legal rights or releasing the investigator(s) or involved institution(s) from their legal and professional responsibilities.

Title of the study: Usability of AWS identity based policies with "read," "write" and "execute" actions.

I have read the information presented in the information letter about a study being conducted by Boyun Zhang, under the supervision of Mahesh Tripunitara of the Department of Electrical and Computer Engineering at the University of Waterloo. I have had the opportunity to ask any questions related to this study, to receive satisfactory answers to my questions, and any additional details I wanted.

I am aware that my session is going to be recorded to ensure an accurate record of my responses and for later analysis and data collection. I was informed that I may withdraw my consent before and during the study without penalty by advising the researcher. However, I will not be able to request withdrawal of the data after study participation is completed. This project has been reviewed by, and received ethics clearance through, the Office of Research Ethics at the University of Waterloo.

I was informed that if I have any comments or concerns resulting from my participation in this study, I may contact the Director, Office of Research Ethics at 1-519-888-4567 ext. 36005 or ore-ceo@uwaterloo.ca.

☐ I am aware than my session is going to be screen captured and audio recorded in order to document any mistakes the participant makes for later analysis and data collection.

☐ I give permission for the use of anonymous quotations in any thesis or publication that comes from this research.

I agree of my own free will to participate in the study.

Participants name: _____

Participants' signature: _____ Date: _____

Researchers/Witness signature: _____ Date: _____

Figure 4.5: Study Acknowledgement in Consent Letter

4.4.6 Limitation

We have chosen the order of the task based on what we believe to have the increasing difficulty.This could also lead to effects due to the ordering that we have not considered. In this study, we also have not considered the seasoned technicians with previous experience of AWS identity-based policy, but only the students familiar or have experience on the operations of the file system with traditional UNIX permissions and are conversant with UNIX permissions in file systems, we believe that the population of the participants is still appropriate given the thesis that we seek to prove.

Chapter 5

Data Analysis

In this chapter, the data analysis result is presented based on the collected data on the small-scale human participant study.

Two participants were treated as pilot participants, they were used to optimize the manner in which the user study we conducted to the rest of the participants. Each of them was participated in different group which was divided in Chapter 4. The one who took part in the user study with currently used AWS identity-based policy had difficulty of understanding the structure of appendix and another had confusion of the mapping procedure from the AWS service resource to the file system with traditional UNIX permissions. Thus the training material was changed to mention these confusions as mistakes to be avoided.

We present the results with accuracy and time for the task completion which are the two main criteria we consider. The graphs are presented for the means of time and the proportions of correct completion. Considered the statistical significance, APA format is considered to be used in the presentation[4], the APA style guide details precise requirements for citing the results of statistical tests, as well as getting the basic format right, the presentation also watch out for punctuation, the placing of brackets, italicisation. We select non parametric test instead of t-test, due to the fact that we are not sure whether the collected data resemble a normal distribution. We used the Mann Whitney U test for statistical significance of interval variables.[18] While for the non-interval variables, we used one sided Fisher's exact test for the accurate rate for the task.

5.1 Time and Accuracy

In this section, each criteria is considered in two types, first is for the study-wide result and another result is considered for each specific task. For each non-parametric test, we adopt a threshold of 0.05 for the p value to regard as the difference of statistically significance. If the p value is smaller than 0.001, we report as "<0.001" for the result. Our null hypothesis for both time and accuracy is that the performance of AWS currently-used identity-based policy is no better than the AWS identity-based policy with "read", "write" and "execute" actions. Which also means that the participants take at least as much time, and are at most as the same accuracy.

5.1.1 Time

Study-Wide Results

For each group of participants, we had 10 participants to finish 3 tasks for the user study. That is, our result included the time taken for each participant to accurately finished the tasks. For the participants from the group of AWS identity-based policy with "read", "write" and "execute" actions, it took 9:37 (minutes : seconds) to finish a task, while for the participants from the group of AWS currently-used identity based policy, the mean time was 7:12. Then the Mann Whitney U Test was launched to check the statistical significance of difference. The result was: $N = 20, U = 26.5, p_{result} = 0.041$. Based on the result that the p_{result} is lower than the threshold we set before, we reject the hypothesis and conclude that the AWS currently-used identity-based policy perform better than the AWS identity-based policy with "read", "write" and "execute" actions from the standpoint of mean time to complete a task.

Specific Task Result

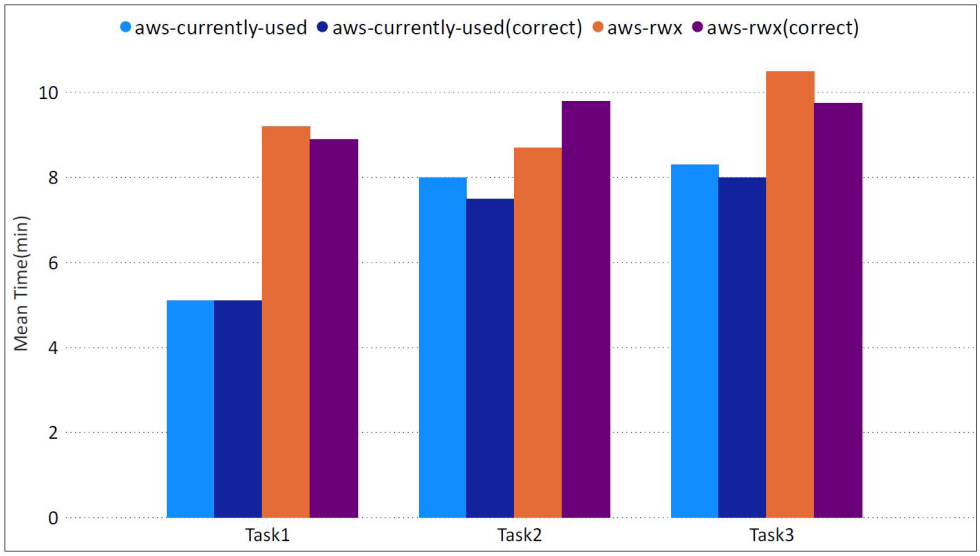


Figure 5.1: Mean Time of Completion

Figure 5.1 presented the result of mean time for each task being completed by participants, and also illustrated the mean time of the task completion which is regard as correct. Another sets of Mann Whitney U Tests were launched to check the statistical significance of difference for each task.

- For Task 1, AWS identity-based policy with "read", "write" and "execute" actions spend significantly more time than the AWS currently-used identity-based policy group:
  $N = 19, U = 16.5, p_{result} = 0.011$.
- For Task 2, the result of Mann Whitney U Test is hard to reject the null hypothesis:
  $N = 11, U = 6.5, p_{result} = 0.07$.
- For Task 3, the result of Mann Whitney U Test is hard to reject the null hypothesis:
  $N = 10, U = 9, p_{result} = 0.159$.

According to the result above for each task, we strongly reject the hypothesis in Task 1, and establish that the AWS currently-used identity-based policy perform better than the AWS identity-based policy with "read", "write" and "execute" actions. While for task 2 and task 3, it's hard for us to reject the null hypothesis based on the time of accurately finish the task.

5.1.2 Accuracy

Study-Wide Results

For each group of the participants, we figured out the proportion of $10 \times 3 = 30$ tasks for each group of participants to accurately finished. For the participants from the group of AWS identity-based policy with "read", "write" and "execute" actions, 18/30 was correctly completed, and 21/30 for the currently-used AWS identity based policy group. Then the Mann Whitney U Test was launched to check the statistical significance of difference for the proportion of each participants correctly finishing the three tasks: $N = 20, U = 38, p_{result} = 0.109$. Hence, the null hypothesis cannot be rejected and it's hard for us to conclude that the AWS identity-based policy with "read", "write" and "execute" actions performs significantly better than the currently-used AWS identity-based policy based on the accuracy.

Specific Task Result

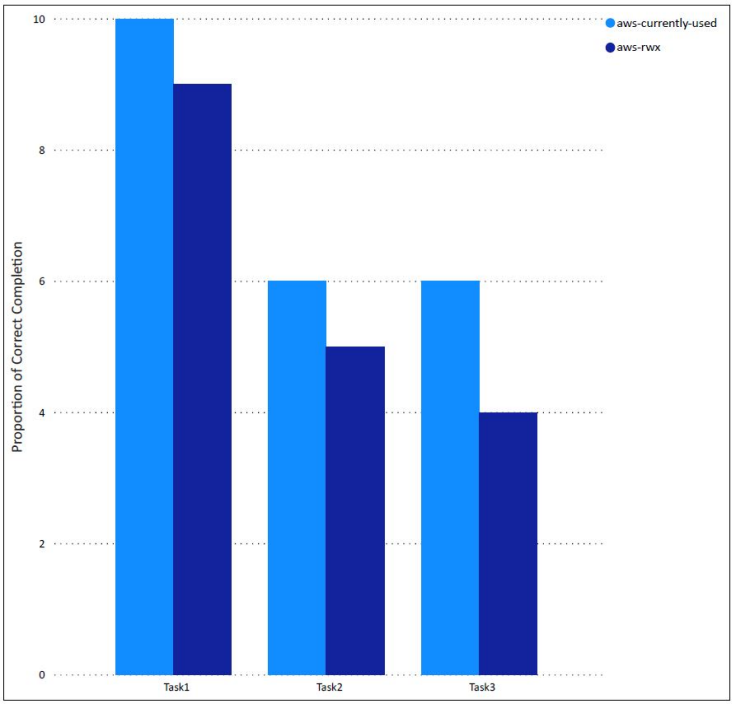


Figure 5.2: Accuracy Proportion

Figure 5.2 presented the accuracy proportion for each task. To figure out the statistical

significance of difference, a set of one sided Fisher's exact test were launched for each task.

- For Task 1, the result of Fisher's Exact Test is hard to reject the null hypothesis:
  AWS identity-based policy with "read", "write" and "execute" actions: 9/10,
  AWS currently-used identity-based policy: 10/10,
  $p_{result} = 1.0$.
- For Task 2, the result of Fisher's Exact Test is hard to reject the null hypothesis:
  AWS identity-based policy with "read", "write" and "execute" actions: 5/10,
  AWS currently-used identity-based policy: 6/10,
  $p_{result} = 0.757$.
- For Task 3, the result of Fisher's Exact Test is hard to reject the null hypothesis:
  AWS identity-based policy with "read", "write" and "execute" actions: 4/10,
  AWS currently-used identity-based policy: 6/10,
  $p_{result} = 0.757$.

From the result above, we cannot reject the null hypothesis for all three tasks based on the accuracy.

5.2 Usability of default directory permission and improvement

According to the small-scale human participants study result, neglect considering the directory permission happened frequently for participants. According to the concept of file system and traditional UNIX permissions we discussed in Chapter 2, the default permission of a directory is set by the command **umask**, while the user is no needed to manually define the permission for directory creation which caused the permission omission in the study.

After analyzing the mistakes occurred by the participants of AWS identity-based policy with "read", "write" and "execute" actions group. All the occurred problems are concluded as below:

- 4 of the total 10 participants have occurred the permission omission problem in one or more tasks, that they ignored considering the permission for the parent directory of the target file.

- 4 of the 10 participants did not have a solid understanding of the traditional UNIX permission and had an incorrect using of permission. For example, participants have given "read" permission to the 'edit-file" action.
- 2 of the 10 participants have provided extra permissions for the actions.

8 of 10 participants have occurred one or more problems mentioned above, and 2 participants have correctly finished all the three tasks. The permission omission and the incorrect using of permission are the two biggest problems throughout the participants.

Therefore we have some suggestions for the future user study procedure, the default directory permission can be given following the default permission of the **umask** setting, while the user is able to manually modify the default permission after the directory is created which can improved the result accuracy.

5.3 Conclusion

Based on the above analysis for time and accuracy of the user study, the null hypothesis can be rejected in Task 1, while it's hard to yield the significant difference in task 2 and task 3. For task 2 and task 3, the participants of the AWS identity-based policy with "read", "write" and "execute" actions group have more procedures to consider not only the syntax of the policy but also the mapping from the AWS service resource to file system with traditional UNIX permissions which have spent more times. And some participants from this group did not have a solid understanding of traditional UNIX permissions after the training, which caused the accuracy of this group did not perform significantly better than another group. The under-privilege is the most common mistakes while the participants ignored considering the permission of all the parent directories for the target file. Overall, the result of the user study is meeting our expectation.

Chapter 6

Conclusions and Future Work

In this thesis we have conducted a detailed observation of the current syntax design on the AWS identity-based policy. Based on the legacy of file systems with traditional UNIX permissions using three actions: "read", "write" and "execute" on the resource management, we have made the modification on the syntax of the AWS identity-based policy in order to help AWS community developer easier to adhere least-privilege and let user more convenient on access control. The compiler has conducted in order to translate and provide all possible sets of actions based on the optimized policy in order to bring convenience for user to manage the service access. We have presented the design and launched a small-scale human participant study to made an observation on our hypothesis. The study result establish that in some scenarios, AWS currently-used identity-based policy perform better than the AWS identity-based policy with "read", "write" and "execute" actions. But it is still understandable and modification on the syntax of the AWS identity-based policy can help AWS community developer easier to adhere least-privilege and let user more convenient on access control.

There are amounts of topics and research areas for future works. The scope of the study participants can be expanded to a larger field-study with experienced AWS community developers as to whether there is a difference to them between two types of policies. If the modification and the hypothesis is proved to be feasible, the project can be kept forwarding to automate the transformation from the optimized policies with "read", "write" and "execute" actions to the currently-used AWS identity-based policy syntax, and the policies can be deployed in conjunction with the cloud application. Besides, another interesting piece of work is to expand this syntax of policy to all the other services in AWS, even other cloud providers can be examined in the same manner as we do in this work.

References

[1] Aaron Blankstein and Michael J Freedman. Automating isolation and least privilege in web services. *Security and Privacy (SP), 2014 IEEE Symposium on.IEEE*, pages 133–148, 2014.

[2] Dick Grune and Ceriel J.H. Jacobs. *Modern Compiler Design*, volume 3. 2012.

[3] Robert Love. *Linux System Programming*, volume 1. 2013.

[4] Nadim Nachar. A test for assessing whether two independent samples come from the same distribution. 2008.

[5] Mathew Sanders and Chuan Yue. Automated software engineering least privileges in cloud-basedweb services. *Proceedings of the fifth ACM/IEEE Workshop on Hot Topics in Web Systems and Technologies*, 3:1–6, 2017.

[6] Ravi Sandhu, Venkata Bhamidipati, and Qamar Munawer. The arbac97 model for role-based administration of roles. *ACM Transactions on Information and System Security (TISSEC)*, 3(1):106–109, 1999.

[7] Ravi S Sandhu, Edward J Coyne, Hal L Feinstein, and Charles E Youman. Role-based access control models. *Computer*, 29(2):38–47, 1996.

[8] Amazon Web Service. Amazon elastic transcoder. `https://docs.aws.amazon.com/elastictranscoder/latest/developerguide/elastictranscoder-dg.pdf`, 2012. Accessed: 2020-09-30.

[9] Amazon Web Service. Identity-based policies and resource-based policies. `https://aws.amazon.com/iam/`, 2013. Accessed: 2020-06-23.

[10] Amazon Web Service. Identity-based policies and resource-based policies. `https://docs.aws.amazon.com/IAM/latest/UserGuide/access_policies_identity-vs-resource.html`, 2014. Accessed: 2020-08-13.

[11] Amazon Web Service. Aws lambda – run code without thinking about servers; pay only for the compute time you consume. https://aws.amazon.com/dynamodb/lambda, 2016. Accessed: 2020-05-21.

[12] Amazon Web Service. Aws sdk for python reference. https://boto3.amazonaws.com/v1/documentation/api/latest/reference/services/elastictranscoder.html, 2016. Accessed: 2020-09-21.

[13] Amazon Web Service. Amazon cloudwatch observability of your aws resources and applications on aws and on-premises. https://aws.amazon.com/dynamodb/cloudwatch, 2018. Accessed: 2020-07-16.

[14] Amazon Web Service. Editing iam policy. https://docs.amazonaws.cn/en_us/IAM/latest/UserGuide/access_policies_manage-edit.html, 2018. Accessed: 2020-08-11.

[15] Amazon Web Service. How to create an aws iam policy to grant aws lambda access to an amazon dynamodb table. https://aws.amazon.com/cn/blogs/security/how-to-create-an-aws-iam-policy-to-grant-aws-lambda-access/, 2018. Accessed: 2020-07-24.

[16] Amazon Web Service. Amazon dynamodb – fast and flexible nosql database service for any scale. https://aws.amazon.com/dynamodb/, 2020. Accessed: 2020-06-11.

[17] Amazon Web Service. Amazon s3 – object storage built to store and retrieve any amount of data from anywhere. https://aws.amazon.com/dynamodb/S3, 2020. Accessed: 2020-10-21.

[18] Rosie Shier. The mann-whitney u test. https://www.statstutor.ac.uk/resources/uploaded/mannwhitney.pdf, 2004. Accessed: 2020-10-21.

[19] Wenjie Tu, Haibing Guan, and Yingcai Bai. Enhancing access control function in linux file system. *Computer Applications and Software*, 23:117–119, 2006.

[20] Jaideep Vaidya, Vijayalakshmi Atluri, and Janice Warner. Roleminer:mining roles using subset enumeration. *In Proceedings of the 13th ACM conference on Computer and communications security(ACM)*, pages 144–153, 2006.

[21] Yongzheng Wu, Jun Sun, and Yang Liu. Automatically partition into least privilege components using dynamic data dependency analysis. *IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2013.

[22] S. Yu, Wang, Ren C., and Lou K. Achieving secure, scalable, and fine-grained data access control in cloud computing. *INFOCOM*, pages 534–542, 2010.

[23] Zahoor, Perrin E., and Bouchami O. A cloud based authorization framework with trust and temporal aspects. *10th IEEE International Conference on Collaborative Computing: Networking, Applications and Worksharing, CollaborateCom*, pages 285–294, 2014.

[24] Gaoshou Zhai and Yaodong Li. Analysis and study of security mechanisms inside linux kernel. In *International Conference on Security Technology*, pages 194–201, 2008.

[25] Gaoshou Zhai, Jie Zeng, Miaoxia Ma, and Liang Zhang. Implementation and automatic testing for security enhancement of linux based on least privilege. *IEEE CS Proceedings of The 2nd International Conference on Information Security and Assurance (ISA 2008)*, pages 181–186, 2008.