Linköping University | Department of Computer and Information Science
Master's thesis, 30 ECTS | Datateknik
23 | LIU-IDA/LITH-EX-A--23/058--SE

Increasing Efficiency and Scalability in AWS IAM by Leveraging an Entity-centric Attribute- & Role-based Access Control (EARBAC) Model

Pontus Jönrup
Rasmus Karlsson

Supervisor : Mohammad Borhani
Examiner : Andrei Gurtov

External supervisor : Thomas Eckestad

LINKÖPING UNIVERSITY

Linköpings universitet
SE–581 83 Linköping
+46 13 28 10 00 , www.liu.se

Upphovsrätt

Detta dokument hålls tillgängligt på Internet - eller dess framtida ersättare - under 25 år från publiceringsdatum under förutsättning att inga extraordinära omständigheter uppstår.

Tillgång till dokumentet innebär tillstånd för var och en att läsa, ladda ner, skriva ut enstaka kopior för enskilt bruk och att använda det oförändrat för ickekommersiell forskning och för undervisning. Överföring av upphovsrätten vid en senare tidpunkt kan inte upphäva detta tillstånd. All annan användning av dokumentet kräver upphovsmannens medgivande. För att garantera äktheten, säkerheten och tillgängligheten finns lösningar av teknisk och administrativ art.

Upphovsmannens ideella rätt innefattar rätt att bli nämnd som upphovsman i den omfattning som god sed kräver vid användning av dokumentet på ovan beskrivna sätt samt skydd mot att dokumentet ändras eller presenteras i sådan form eller i sådant sammanhang som är kränkande för upphovsmannens litterära eller konstnärliga anseende eller egenart.

För ytterligare information om Linköping University Electronic Press se förlagets hemsida `http://www.ep.liu.se/`.

Copyright

The publishers will keep this document online on the Internet - or its possible replacement - for a period of 25 years starting from the date of publication barring exceptional circumstances.

The online availability of the document implies permanent permission for anyone to read, to download, or to print out single copies for his/hers own use and to use it unchanged for non-commercial research and educational purpose. Subsequent transfers of copyright cannot revoke this permission. All other uses of the document are conditional upon the consent of the copyright owner. The publisher has taken technical and administrative measures to assure authenticity, security and accessibility.

According to intellectual property law the author has the right to be mentioned when his/her work is accessed as described above and to be protected against infringement.

For additional information about the Linköping University Electronic Press and its procedures for publication and for assurance of document integrity, please refer to its www home page: `http://www.ep.liu.se/`.

© Pontus Jönrup
Rasmus Karlsson

Abstract

Cloud computing is becoming increasingly popular among all types of companies due to its inherent benefits. However, because of its infrastructure, it might be difficult to manage access rights between users and resources. To address these difficulties, Amazon Web Services (AWS) provides Identity and Access Management (IAM) and features that support the use of different access control models, for example, Role-based Access Control (RBAC) and Attribute-based Access Control (ABAC). Access control models are used for authorisation within systems to decide who gets access to what. Therefore, to determine what constitutes an efficient (the average time it takes to perform a task in AWS IAM) and secure access control model, a thorough study of background material and related work was conducted. Through this study, it was found that RBAC lacked scalability whilst ABAC lacked administrative capabilities. It was also found that flexibility and scalability were two important factors when designing access control models. Furthermore, by conducting a survey and designing an access control model for AWS through various iterations, a new access control model called Entity-centric Attribute- & Role-based Access Control (EARBAC) was developed. In an experiment comparing it with the RBAC model, the EARBAC model was found to be both efficient and secure, in addition to its flexibility and scalability. Furthermore, EARBAC was also found to be 27% faster than RBAC in AWS IAM. These results suggest that the model is useful when developing cloud infrastructures in AWS.

Acknowledgments

We would like to express our sincere gratitude towards our supervisor at Linköping University, Mohammad Borhani, for always providing us with guidance, support, and expertise throughout this thesis. Furthermore, we would also like to extend our sincere gratitude towards our examiner, Andrei Gurtov, for his time, dedication, and constructive evaluation.

We would also like to extend our sincere gratitude towards the accommodating people at NIRA Dynamics who helped us with our work and allowed us to use their facilities and services. Special thanks to our supervisors at NIRA Dynamics, Thomas Eckestad and Roger Wahlström, who provided us with professional insight and knowledge. Finally, we would also like to thank all the employees who participated in our survey and experimentation. Without their help, we would not have been able to conduct our work and the finished result would probably not have been as good.

Contents

Abstract iii

Acknowledgments iv

Contents v

List of Figures vii

List of Tables viii

1 Introduction 1
- 1.1 Motivation . . . 1
- 1.2 Aim . . . 2
- 1.3 Research Questions . . . 2
- 1.4 Approach . . . 2
- 1.5 Delimitations . . . 3
- 1.6 Keywords . . . 3

2 Background 4
- 2.1 Cloud Computing . . . 4
- 2.2 Amazon Web Services . . . 5
- 2.3 Authorisation . . . 6
- 2.4 Principle of Least Privilege . . . 6
- 2.5 Access Control . . . 6
  - 2.5.1 Mandatory Access Control . . . 7
  - 2.5.2 Discretionary Access Control . . . 7
  - 2.5.3 Bell-LaPadula and Biba . . . 7
  - 2.5.4 Role-based Access Control . . . 8
  - 2.5.5 Attribute-based Access Control . . . 8
- 2.6 Identity and Access Management in Amazon Web Services . . . 9
  - 2.6.1 Role-based Access Control in Amazon Web Services Identity and Access Management . . . 13
  - 2.6.2 Attribute-based Access Control in AWS IAM . . . 14

3 Related Work 16
- 3.1 Related Work Pertaining to Role-based Access Control . . . 16
  - 3.1.1 Formal Definition, Feature Expansion, and Evaluation Specification . . . 16
  - 3.1.2 Workflows and Tasks for Flexibility . . . 17
  - 3.1.3 Attribute-inspired Role-based Access Control variations . . . 18
- 3.2 Related Work Pertaining to Attribute-based Access Control . . . 19
  - 3.2.1 Attribute-based Access Control for Scalability . . . 19
  - 3.2.2 Attribute-based Access Control in Cloud . . . 20
- 3.3 General Related Work . . . 21

4 Method **22**
4.1 Survey Method . . . 22
4.2 Proposed Access Control Model . . . 23
4.2.1 Iteration 1: Attributes . . . 24
4.2.2 Iteration 2: Roles . . . 27
4.2.3 Iteration 3: Access Management . . . 29
4.3 Experimentation Method . . . 31
4.3.1 Scoping . . . 31
Goal Definition . . . 32
4.3.2 Planning . . . 32
Context Selection . . . 32
Hypothesis Formulation . . . 33
Variables Selection . . . 33
Selection of Subjects . . . 34
Experiment Design . . . 34
Instrumentation . . . 34
Validity Evaluation . . . 35

5 Results **38**
5.1 Survey Results . . . 38
5.2 Entity-centric Attribute- & Role-based Access Control Model . . . 40
5.3 Experimentation Results . . . 42

6 Experimentation Evaluation **45**

7 Discussion **47**
7.1 Method Discussion . . . 47
7.2 The Work in a Wider Context . . . 49

8 Conclusion **50**

A Survey Questions **52**

B Experimentation Guidelines **54**
B.1 Guidelines for experimentation with Role-based Access Control . . . 54
B.2 Guidelines for experimentation with the Proposed Model . . . 56

C Experimentation Results (complete tables) **59**

D Parameter/NotParameter Table **60**

E JSON Code **61**
E.1 General . . . 61
E.2 Experimentation - Role-based Access Control . . . 62
E.3 Experimentation - Proposed Model . . . 65

Bibliography **69**

List of Figures

2.1 Structure of an AWS Organisation 5
2.2 Structure of a JSON Policy in AWS IAM 9
2.3 Illustration of active and restrictive policies. 11
2.4 To the left is an illustration of an Inline policy. To the right is an illustration of a Resource-based policy. 12
2.5 Three Venn diagrams illustrating the use of restrictive policies. Check marks with brighter colours are evaluated as allowed actions. 12
2.6 Illustration of how an RBAC model could be set up inside of AWS IAM. 13
2.7 Illustration of how an ABAC model could be set up inside of AWS IAM. 14

5.1 Pie chart illustrating the results of the question stated in the figure 39
5.2 Illustration of the EARBAC model within AWS IAM 41
5.3 Chart illustrating the timings for the RBAC model 42
5.4 Chart illustrating the timings for the EARBAC model 42

List of Tables

4.1 Table showing a set of tasks each subject will perform during experimentation . . 34

5.1 Table consisting of the most relevant survey questions and their corresponding answers 39
5.2 Table consisting of the results from the complexity measurements 43
5.3 Table consisting of the results from the risk measurements 43
5.4 Table consisting of the subjects' feedback from the experiment tasks 43
5.5 Table consisting of the average time it took for each subject to complete the tasks in RBAC and EARBAC 44

A.1 General questions 52
A.2 Questions pertaining to Roles in AWS 52
A.3 Questions pertaining to Resources / Services in AWS 53
A.4 Questions pertaining to the general Workflow in AWS 53

C.1 Table consisting of the results from the experiment for the RBAC model 59
C.2 Table consisting of the results from the experiment for EARBAC 59

D.1 Table of parameters and their combination of effects 60

List of Policies

2.1 General JSON Policy Code 10
2.2 Specific JSON Policy Code 10
4.3 Identity-based policy for S3 resources (Public Read(List)) 24
4.4 Identity-based policy for S3 resources (only users with correct tag) 25
4.5 Identity-based policy for S3 resources (using a naming standard) 25
4.6 Identity-/Resource-based policy for S3 resources (no naming standard) 26
4.7 Trust Policy for an IAM Role 28
4.8 Identity-based policy for a Developer Role (Resource Specific) 28
4.9 Identity-/Resource-based policy for all three Roles 29
4.10 Identity-based policy for Ensuring No-Read-Up and No-Update-Up 30
4.11 Identity-based policy for a New AccessManager Role 31
5.12 Updated Trust Policy for an IAM Role 40
E.13 Identity-based Policy for a Developer Role (S3, EC2, EKS, Other Resources) . . 62
E.14 Experimentation Policy 1 for Role-based Access Control 62
E.15 Experimentation Policy 2 for Role-based Access Control 63
E.16 Experimentation Policy 3 for Role-based Access Control 63
E.17 Experimentation Policy 4 for Role-based Access Control 64
E.18 Experimentation Policy 5 for Role-based Access Control 64
E.19 Experimentation Policy 6 for Role-based Access Control 65
E.20 Experimentation Policy 1 for Proposed Model 65
E.21 Experimentation Policy 2 for Proposed Model 66
E.22 Experimentation Policy 3 for Proposed Model 66
E.23 Experimentation Policy 4 for Proposed Model 67
E.24 Experimentation Policy 5 for Proposed Model 68

1 Introduction

This chapter introduces the main problem and motivation of this thesis. Furthermore, the research questions are presented as well as the intended approach.

1.1 Motivation

Amazon Web Services (AWS) is a subsidiary of Amazon that provides *cloud computing* solutions to users in nearly one hundred geographical locations worldwide and is expanding its business to reach even more customers [26]. Some of their largest and most notable customers include NASA[1], Epic Games[2], and Netflix[3], among others.

A cloud computing solution means that a customer gets remote access to physical hardware, from a cloud vendor such as AWS, to run their software on. To limit access from unauthorised people and enable access to certain ones, *access control* methods are in place to ensure this. Outside of login credentials, users usually need certain privileges or other access rights even inside the system. This is to protect sensitive information and critical infrastructure from potential attacks that could potentially get past the first line of defence, but also from actual users without a certain status, such as administrator, team leader, etc.

AWS *Identity and Access Management* (IAM) provides a toolbox for implementing access control to resources in AWS[4]. The access control in AWS IAM is based upon *policies* that enable access to certain resources. These policies can then be associated with developer *roles* (e.g. guest, user, admin), which are assigned manually. When systems using AWS IAM grow, as a result of adding more services and infrastructure, there has to be a clear and simple way to set up access control for new resources. If careful analysis of different roles for developers is not conducted before creating a solution, it is too easy to lock out and block these developers from resources that they need in their daily work. Thus, the access management setup will be affected by at least: best practices, customer agreements, and way-of-working. It is also crucial that the setup is kept as simple as possible because, if a developer is hindered by the access control in an environment where the data is only accessible for a short amount of

[1] https://aws.amazon.com/blogs/publicsector/perseverance-lands-mars-cloud-ready-explore/?hp=tile&tile=customerstories&refid=cr_card

[2] https://aws.amazon.com/solutions/case-studies/innovators/epic-games/?did=cr_card&trk=cr_card

[3] https://aws.amazon.com/solutions/case-studies/innovators/netflix/

[4] https://aws.amazon.com/premiumsupport/knowledge-center/iam-intro/

time, they might have to be assigned a role with overqualified access; this would be considered a security breach. Therefore, a model that takes both developer efficiency and security into account would need to be applied to the system to prevent developers from gaining unauthorised access to new resources.

Furthermore, according to the Open Web Application Security Project, the most common vulnerability in web applications is *broken access control*[5]. If access control in a web application or infrastructure is managed incorrectly, it can lead to unauthorised actors gaining access to information or other resources outside of their intended privilege. However, it is a difficult challenge to implement access control correctly as there are several different access control mechanisms with their own pros and cons [15]. Depending on how many or what type of resources a cloud computing service has, one *access control model* may solve the relevant security issues better than another model. This is because their purpose is not just to limit access but also to enable access for *authorised* users. Finally, according to Shaikh and Haider [29], "It is being observed that the users of cloud computing services are not satisfied with the current security mechanism in cloud computing".

The foundation of this thesis is made in collaboration with NIRA Dynamics, a Swedish company specialising in making analytical software for cars and road surfaces. Their insight and knowledge in AWS IAM will support the work in this thesis and during the experimentation and evaluation phase.

1.2 Aim

The goal of this thesis is to introduce and analyse models and strategies for how to best strike a balance between security and developer efficiency in AWS IAM. The solutions have to be extendable and the key concepts should be clear and perceivable for anyone working with similar systems.

1.3 Research Questions

This thesis aims to answer the following research questions, where efficiency means the average time it takes to perform a task in AWS IAM:

1. What counts as an efficient security policy strategy?
2. How can a balance be found between security and developer efficiency utilising the strategy found in RQ_1 for AWS IAM?
3. How can AWS IAM be used to create an access control model that uses the advantages of both Role-based Access Control and Attribute-based Access Control?

1.4 Approach

The research for this thesis consists of reading material from scientific literature, conferences, and instructions from web pages such as IEEE Xplore[6], Springer[7], Google Scholar[8] and the library of Linköping University[9]. Information from these sources helps answer the research questions presented in this thesis, and research topics include the following, but are not limited to, AWS IAM, access control, policy distribution, and relations between security and developer efficiency. A Proof of Concept of a proposed access control model will be implemented and evaluated.

[5]https://owasp.org/www-project-top-ten/
[6]https://ieeexplore.ieee.org/Xplore/home.jsp
[7]https://link.springer.com/
[8]https://scholar.google.com/
[9]https://liu.se/en/library

1.5 Delimitations

There are access control models that manage both *authentication* and *authorisation*. However, in the context of this thesis, a user is already logged in to AWS and identified (authenticated), which means that this thesis does not research the subject matter of authentication, therefore focusing only on the authorisation. While most of the presented topics on access control can be applied to cloud computing services such as AWS, Microsoft Azure[10], Google Cloud[11], etc., this thesis only applies them to AWS. This thesis does not consider encryption when researching different access control models and uses AWS built-in objects.

1.6 Keywords

AWS Amazon Web Services

S3 Simple Storage Service

IAM Identity and Access Management

RBAC Role-based Access Control

ABAC Attribute-based Access Control

EARBAC Entity-centric Attribute- & Role-based Access Control

[10]https://azure.microsoft.com
[11]https://cloud.google.com/

2 Background

In this chapter, the basics of cloud computing, Amazon Web Services, and access control will be presented, as well as previous models that have laid the foundation for modern access control. AWS IAM and its concepts will also be introduced to present a clear overview of the environment in which this thesis is conducted.

2.1 Cloud Computing

The term cloud computing was first used by Challeppa [23] in 1997, and it refers to when you run software in the cloud, compared to running it locally. Cloud computing can offer services such as data storage, video streaming, different kinds of processing, and other services. The idea behind cloud computing is that a vendor enables remote access to their hardware for users to run software, access services, build platforms or their own services, etc., from anywhere in the world [3, 28]. The vendor has total control over the hardware, but the user has some control over the services, depending on the type of service model the cloud vendor offers. A key feature of cloud computing is *elasticity*, the ability to dynamically increase or decrease the resources needed for cloud computing operations [18, 28]. This feature enables the users of a cloud computing service to scale their operations up or down, depending on how much they need at the moment, or how much they are planning on using. The cloud vendors' hardware run virtual machines which in turn run the customers' software. The virtual machines run in the vendors' data centres, and allocating new resources to customers means finding an open spot on physical machines to place another virtual machine on.

Furthermore, there are three primary cloud service models: *Software-as-a-service* (SaaS), *Platform-as-a-Service* (PaaS), and *Infrastructure-as-a-Service* (IaaS) [28, 20]. SaaS is a service that users can access without having control over, or having to worry about, the underlying hardware, operating system, or virtualisation. An example of this would be online word processors, where you can write and save your progress remotely. PaaS provides a software service as well but is primarily used by developers, testers, and administrators to develop applications. While the vendors still have full control, PaaS offers more control to the users. IaaS is the most fundamental part of cloud computing as the other two service models are built upon it. IaaS provides an infrastructure for developers and other users to run their platform and applications on and they have the most control over the cloud in this service

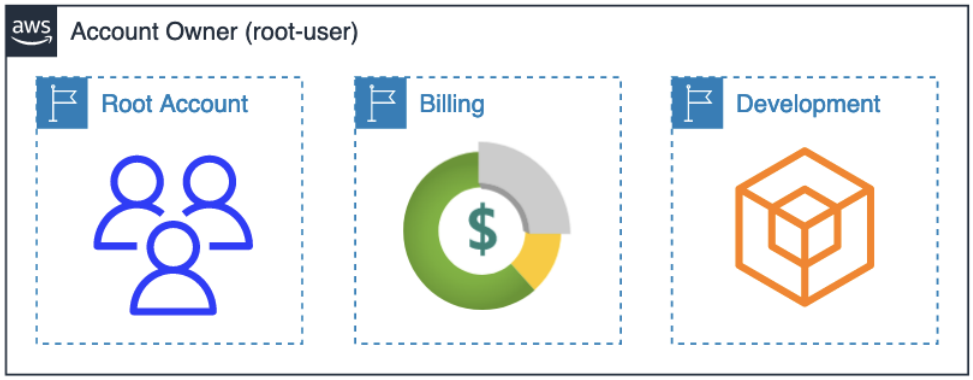

Figure 2.1: Structure of an AWS Organisation

model. They can run arbitrary programs and operating systems and they have access to other aspects as well, such as storage, networking, processing, and main computing resources.

Outside of the three primary cloud service models, Monshizadeh et al. [20] introduce a fourth service model, catered towards mobile operators, called *Telecommunications network as a Service* (TaaS). According to the authors, there are network operators called *Mobile Virtual Network Operators* (MVNOs) that are hosted by a physical mobile network. Instead of having its own support system, an MVNO could become a customer of a Mobile Virtual Network Enabler, and these services could be implemented using cloud computing. In another paper, by Gurtov et al. [19], including Monshizadeh, the authors review TaaS in regards to sharing physical and virtual resources over a mobile network running on cloud computing. However, due to security concerns, they conclude that the existing frameworks they used for implementing TaaS need to be revised before being ready for deployment. Gurtov et al. [1] (another set of co-authors) introduce *Software-Defined Mobile Networks* which, similarly to MVNOs, use virtualisation to improve the capacity scaling and adaptability of the architecture in mobile networks. The idea is that it should better fit varying and diverse traffic demands, and according to the authors, mobile service providers turn towards cloud computing for solutions in this area.

2.2 Amazon Web Services

In 2006, Amazon created AWS which would specialise in creating cloud computing solutions. Anyone can buy their services, but companies who want to utilise the advantages of cloud computing benefit the most from AWS[1]. By offering a wide variety of IT infrastructure services on a pay-as-you-go model, they allow businesses to mitigate the risk, and reduce the time and capital needed, to build their own infrastructure[2]. Some of the areas in which AWS supplies services are compute clouds, storage solutions, databases, and networking & content delivery[3], but AWS also offers services such as *Organizations*[4] and IAM[5]. The AWS Organization service lets the account owner create an organisation where they can create and manage multiple accounts simultaneously, see Figure 2.1 above, and then allocate resources as needed. This service can also be used to apply governance policies to accounts and groups within the organisation.

A specific storage solution that AWS offers is the *Simple Storage Service* (S3), which is an object-storing service that puts emphasis on *scalability*, *data availability*, *security*, and *perfor-*

[1]https://aws.amazon.com/getting-started/
[2]https://aws.amazon.com/about-aws/
[3]https://aws.amazon.com/products/
[4]https://aws.amazon.com/organizations/
[5]https://aws.amazon.com/iam/

mance [27]. Data in S3 is stored in a type of resource called *bucket*, and within those buckets, data is stored as *objects*. The object itself can consist of a text file, a photo, a video, etc., and any metadata related to the file. In other words, buckets act as containers for data, and an AWS account can store several of them. For each bucket, an AWS *user* can choose the geographical region where S3 shall store the bucket and everything inside it.

To run conduct cloud computing and run programs, AWS offers several solutions, and their biggest service in this field is Elastic Compute Cloud (EC2)[6] that enables a customer to tailor a virtual machine specifically to their needs and offers elastic cloud computing to meet the customer's demands. Another service by AWS is Elastic Kubernetes Service (EKS)[7] that enables and runs Kubernetes[8] in the cloud provided by AWS.

2.3 Authorisation

According to Fraser [11], authorisation means granting privileges to processes and users. A privileged user, one who has gained privileges through authorisation, has the ability to interact with objects in a system through a set of permissions, which usually consist of the three standard permissions: *read*, *write*, and *execute*. In a standard system, the owner of an object, which is usually the object creator, is authorised all the permissions related to their object and can usually also delegate who and what can interact with it as well. Apart from authorisation, there is also *authentication* and it is important to distinguish these two concepts. While authorisation is used to grant privileges to a user, authentication is used to identify a user, according to Fraser, as well as Saltzer and Schroeder [24].

2.4 Principle of Least Privilege

The *principle of least privilege* is one of the secure design principles created by Saltzer and Schroeder [24]. The idea behind the principle is that programs and users in a system should only operate with the least amount of privileges necessary to do their tasks. As an example, if a user does not need access to a certain resource, the user should not have the privilege to access that resource. This helps reduce the number of privileged operations interacting with each other, reducing the risk of unwanted access. Furthermore, the resulting damages from accidents or errors are limited by this principle, as well as unwanted, improper, or unintentional uses of privileged actions.

In an article, by Jero et al. [16], the authors talk about how they developed an embedded system adhering to the principle of least privilege. They list several challenges involved in creating such a system. Among them are *granularity* which states that privilege must be as fine-grained as possible when it comes to accessing resources and shared data structures. This is to ensure that potential violations of intended usage will be denied. Another challenge is *recovery*, which states that faulty or malicious processes should be able to recover, but in a way that does not compromise the system's security.

2.5 Access Control

Access control has been in place for a long time, using physical sensors and components to ward off sensitive information or resources to unauthorised access in buildings and other locations [9]. The same methods, although a bit physically different, are applied to online infrastructure as well. Access control is the process of securing data, which enables organisations to control who gets access to what[9]. Among other concepts, access control uses autho-

[6] https://aws.amazon.com/ec2/
[7] https://aws.amazon.com/eks/
[8] https://kubernetes.io/
[9] https://www.fortinet.com/resources/cyberglossary/access-control

risation to decide which users get access rights to certain resources. Furthermore, this section will focus on explaining four types of traditional access control mechanisms, as well as explaining their advantages and disadvantages. These models are commonly used today and serve as a good foundation for understanding how modern access control works.

2.5.1 Mandatory Access Control

In *Mandatory Access Control* (MAC), the operating system or security kernel handles the access control and sets the access permissions for objects [15]. A systems administrator sets the access permissions for objects, and that access control is later enforced by the operating system or security kernel. They enable or deny users the right to grant permissions in the file system, and users can not alter this. MAC uses classification labels for the objects in the file system, which means an object can be labelled as different security levels: secret, top secret, or confidential. Not only the objects have this label, but the users and other devices as well, and are also assigned a clearance level which dictates that they have the privilege to access objects of the same level. The classification labels of the users and the objects are set by the security kernel and the credentials. Accessing rights are checked by the security kernel or operating system upon access to determine if the one trying to access should be denied or allowed access. MAC is a secure model when it comes to access control, but it requires considerable planning because the classification labels need to be updated. It is therefore less flexible than some other models. Once the level of security is defined in MAC, the safety level does not adjust in the hierarchy [3]. Because of these aspects, the MAC model's scalability is worse than for other models, and might not be applicable everywhere.

2.5.2 Discretionary Access Control

In *Discretionary Access Control* (DAC), a user who owns an object controls the access to it, hence the *discretionary* part [15]. Authentication in DAC is performed through a login where a user enters their username and password, during which their access rights are performed. Since it is up to the owner of the object to create access control policies, dictating who can access their owned object, DAC provides more *flexibility* than some other models. However, for the same reason, it is also considered less secure. DAC allows for the owner to delegate access privileges of their object to other subjects, effectively enabling object sharing [3]. DAC also allows for access inheritance, user-based permissions, process activity auditing, and administrative rights. However, it cannot constrain the usage of data and its flow and is weak to attacks by unauthorised users who could use malware to gain access to information even without the object owner's permission.

2.5.3 Bell-LaPadula and Biba

The *Bell-LaPadula* (BLP) and *Biba* access control models are old models that are designed to protect the *confidentiality* and *integrity* aspects, respectively [31]. Both models have been used plenty for military purposes, but less so in commercial fields due to their lack of consideration for the security of the channel used for communication. The BLP model works by combining two rules that together prevent confidential information from being accessed by unauthorised subjects: subjects may not read the information at a higher sensitivity level, and subjects may not write information to an object at a lower sensitivity level. In other words, this model enforces the properties *no read up* and *no write down*. The Biba model also works by combining two rules of similar nature: subjects at a given level of integrity must not read data at a lower integrity level, and subjects at a given level of integrity must not write to data at a higher level of integrity. In other words, this model enforces the properties *no read down* and *no write up*.

2.5.4 Role-based Access Control

In 1992, Ferraiolo et al. [10] defined the non-discretionary access control model called *Role-based Access Control* (RBAC) that built upon the previous DAC model used in military systems at the time. This model became well known for its administrative capabilities as well as its support for data integrity. A user gains access rights to objects through roles and permissions in RBAC, and the model categorises the roles into two groups: application/technicality and organisation/business [15]. A role that belongs to the application/technicality group gets permissions that enable access to certain parts of an application, whereas a role within the organisation/business is a combination of several roles from the previous group. This means RBAC can be used for administrative purposes and can be used for larger organisations as well where there are a lot of users and resources to take care of. When assigning permissions to a user, there are three main rules in RBAC that the model follows: role assignment, role authorisation, and permission authorisation. These three rules shape the way access permissions are formed for users and create a secure environment for deciding access rights.

In comparison with the following ABAC model, the RBAC model has been said to give simpler administrative and review capabilities [17, 4, 22, 32], however, the main drawback is that when the environment changes, the roles and permissions do not naturally adapt. These changes have to be made manually and can be a complex task in a complex environment. Furthermore, when a company using RBAC grows, it can quickly become susceptible to something called *role explosion* and *role-permission explosion*, meaning the role structure becomes even more complex when time passes and can result in less scalability if not managed properly [4, 22, 32].

Finally, in a work by Habib et al. [13], the authors compare the RBAC model to the BLP model, using simulated states. In their findings, they conclude that BLP is strictly more restrictive than RBAC. This is because BLP can not simulate all possible states that exist in RBAC. For example, a state in RBAC where no access can be authorised does not exist in BLP.

2.5.5 Attribute-based Access Control

The *Attribute-based Access Control* (ABAC) model was designed by Hu et al. [14] and works with the *attributes* of objects and subjects, as well as other conditions, to decide access to resources. Each object is characterised by its specific attributes and can refer to other objects, such as the owner. Every user of the system using ABAC is also assigned attributes, which can include a name, role and which group the user belongs to. As the system changes, the attributes of subjects may also need to change to match the new environment specifics. ABAC works with policies that decide which attributes to check in order to determine whether a user gets access to a resource or not [15]. This means that the attributes in ABAC shape the policies of access control. Since there are predefined roles and privileges for each user in ABAC, the model becomes efficient and flexible, while also solving authorisation problems.

As mentioned previously, the attributes work on different types of entities, and the types of attributes can be grouped into subject attributes, object attributes, resource attributes, and environmental attributes. Environmental attributes can be for example the time and date that a resource is accessed, helping determine the access rights [3]. The model treats aspects such as classification, authorisation, and transparency, and creates attributes based on the user's request to access resources and what resources the user requires. Due to its structure, ABAC is more scalable and flexible than other models, while also being secure. However, the model might require a large computing *overhead*.

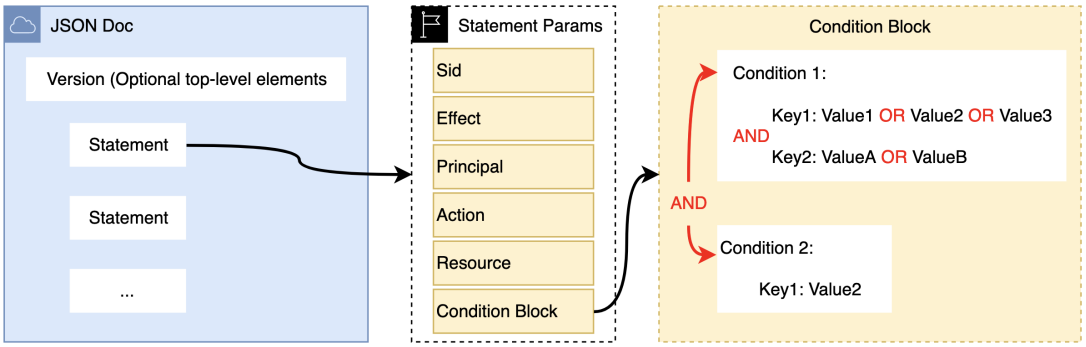

Figure 2.2: Structure of a JSON Policy in AWS IAM

2.6 Identity and Access Management in Amazon Web Services

All information presented in this section is a summarisation of how AWS IAM works[10]. The IAM service in AWS gives the user, or business, a framework that can help apply *fine-grained access control* between other AWS services and resources. By default, however, developers have no access rights. It is the owner of the AWS account that has all the privileges from the start, and it is also up to them to create the initial permissions that give other developers the possibility to work in their corresponding areas. To enable this, the IAM framework introduces different kinds of IAM resources, such as *User*, *User Group*, *Role*, *Tag*, AWS resources, etc. IAM resources are not to be mistaken for normal AWS resources; AWS resources are services such as compute clusters or databases whilst IAM resources refer to data elements that are used to access AWS resources. Furthermore, IAM resources can then be connected and matched with different access control models, such as DAC, MAC, RBAC, and ABAC, which are explained in Section 2.5, using something called *Policies*; In IAM, Policies are also a type of IAM resource. These Policies are the main building blocks of IAM as they can achieve security, scalability, and manageability when utilised accordingly. Additionally, AWS uses something called Amazon Resource Names (ARNs), which are attached to all elements (Users, Roles, Policies, etc.), to uniquely identify resources across all of AWS. Some examples of these ARNs can be seen below in Policy 2.2.

Moving on, Policies are built using JSON files with different parameters depending on what type of policy is being written, more on types later. The most common parameters are *Statement*, *Effect*, *Principal*, *Action*, *Resource*, and *Condition*, however, there are also some semi-optional parameters such as Version and Statement ID (Sid) that can be used for version control and metadata respectively. The JSON file structure is as shown in Figure 2.2, and two code examples are given below in Policy 2.1 and 2.2, where:

- Version is the most important semi-optional top-level parameter that tells the IAM evaluation process which specific version of policy language is to be used when evaluating. To gain all of the latest features, one should always set the Version to "2012-10-17", as this version enables variables. There are also other top-level parameters that can be used.
- Statement is the main container of a policy. There can also be more than one statement in each policy as illustrated in Policy 2.1.
- Effect is, as it sounds, the effect that a Statement will push to AWS. This parameter must be either "Allow" or "Deny", however, one can use multiple Statements with different Effect values if needed.

[10]https://docs.aws.amazon.com/IAM/latest/UserGuide/introduction.html

Policy 2.1 General JSON Policy Code

```
1  {
2    "Version": "2012-10-17",
3    "Statement": {
4      "Effect": "Allow",
5      "Principal": "*",
6      "Action": "*",
7      "Resource": "*",
8      "Condition": {
9        "StringEquals": {
10         "aws:SourceAccount": "111111111111"
11       },
12       "ArnEquals": {...}
13     }
14   },
15   "Statement": {...}
16 }
```

Policy 2.2 Specific JSON Policy Code

```
1  {
2    "Version": "2012-10-17",
3    "Statement": {
4      "Effect": "Allow",
5      "Principal": ["arn:aws:iam::111111111111:user/user-1",
6                    "arn:aws:iam::111111111111:user/user-2"],
7      "Action": ["s3:GetObject",
8                 "s3:PutObject"],
9      "Resource": ["arn:aws:s3:::bucket-1",
10                  "arn:aws:s3:::bucket-2"],
11     "Condition": {...}
12   }
13 }
```

- Principal can be seen as a "calling" entity, also called IAM Entity, that will be denied or allowed some permissions. In IAM, a "calling" entity can be either a User or a Role. One Statement can also have multiple "calling" entities by either using the "wildcard" (*) character, see code line 5 in Policy 2.1, or by using list brackets as input, see code line 5-6 in Policy 2.2, to the Principal parameter.
- Action is the parameter that specifies which action(s) are to be allowed or denied for the specified Principal(s). This parameter can also be switched to NotAction, see usage in Appendix Table D.1. Similarly to the Principle parameter, the Action/NotAction parameter can also have multiple actions by using either "wildcard" (*) or list brackets.
- Resource refers to an AWS Resource or service that some Principal(s) will be able to use Action(s) on. This parameter can also be switched to NotResource, see usage in Appendix Table D.1. The Resource/NotResource parameter can also have multiple resources by using either "wildcard" (*) or list brackets.
- Condition Block is an optional parameter that can be used for multiple purposes and can have one or more conditions within[11], see code line 8-13 in Policy 2.1. The logic of these conditions evaluates using 'AND' and 'OR' as shown in red, furthest to the right, in Figure 2.2.

[11] https://docs.aws.amazon.com/IAM/latest/UserGuide/reference_policies_elements_condition_operators.html

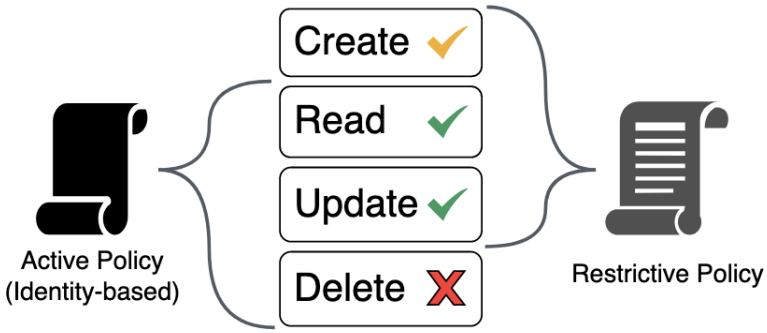

Figure 2.3: Illustration of active and restrictive policies.

Even though policy parameters, excluding Statement, can be written in any order inside of a Statement, a simple way of reading policies would be as shown by the previous bullet points from top to bottom:

> <*Effect*> <*Principal*> to use <*Action*> on <*Resource*> if-and-only-if (iff) <*Condition*>

Therefore, to summarise Policy 2.1:

> <*Allow*> <*all Principals*> to use <*all Actions*> on <*all Resources*> iff <*the aws:SourceAccount is equal to 111111111111* ***AND*** *ArnEquals is equals to some value (not specified here)*>

Also, Policy 2.2 would be summarised to:

> <*Allow*> <*test-user1, test-user2*> to use <*GetObject, PutObject*> on <*S3:::bucket-1, S3:::bucket-2*> iff <*some condition (not specified here)*>

Before constructing these policies, however, some key aspects have to be considered. Firstly, policies have many different types depending on what they are supposed to be attached to, and secondly, policies can be divided into two groups. These groups are not specified in the AWS documentation, but this thesis will call them active- and restrictive policies. The reason for this is that active policies are policies that explicitly allow or deny some action between IAM resources, whilst restrictive policies are policies that restrict the number of actions that an active policy can allow or deny. With this definition, restrictive policies can be seen as higher order compared to active policies and can be used to organise and maintain an organisation on a bigger scale. However, whilst restrictive policies can be seen as higher order, they cannot grant any permissions themselves, it is only the active policies that can actually grant permissions to an IAM Entity. This can be seen by the yellow check mark in Figure 2.3; a green check mark represents a granted action; a red cross represents a denied action. Furthermore, if no restrictive policy is applied, all actions that an active policy grants are allowed. This thesis has grouped the IAM Policy types *Identity-based policy* and *Resource-based policy* into the active policies group because, as per the previous definition, they either allow or deny some action. Meanwhile, the restrictive group involves the IAM Policy types *Permission Boundaries*, *Service Control Policies* (SCPs), and *Session Policies*, as per the same definition.

Regarding the active policies, Identity-based policies further involve two types of policies called *Managed-* and *Inline policies*. Managed policies are policies that can be created once and put into a reusable library to use on multiple IAM Entities. These policies are either AWS managed (AWS themselves manage the policies) or Customer managed (the customer manages their own policies). Inline policies, however, are a bit special as they cannot be stored in a library and are created and attached to only one IAM Entity (one user, multiple

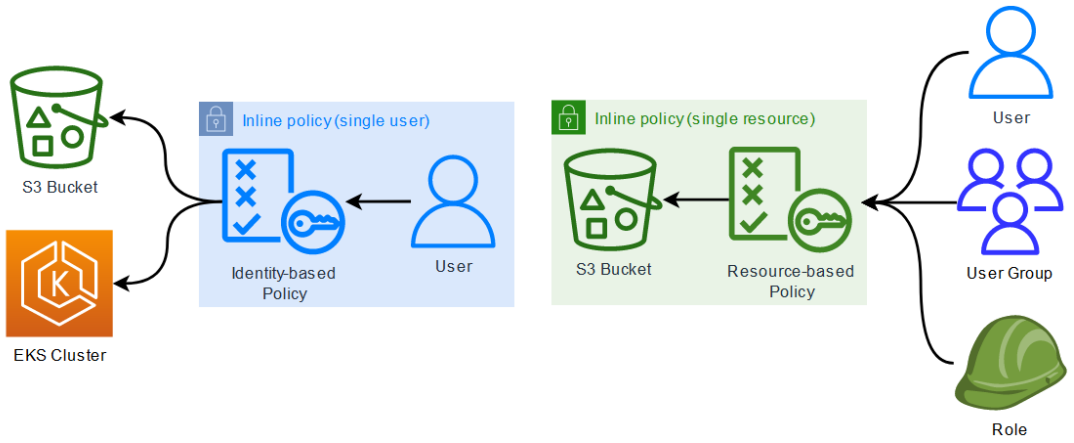

Figure 2.4: To the left is an illustration of an Inline policy. To the right is an illustration of a Resource-based policy.

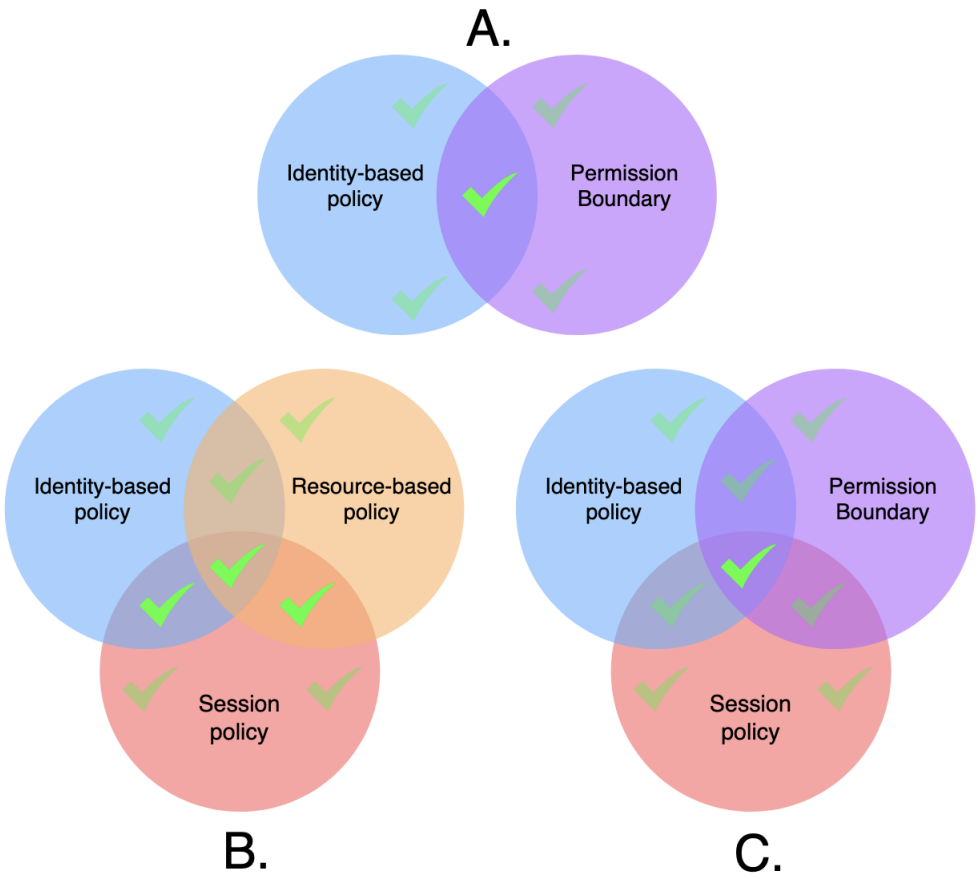

Figure 2.5: Three Venn diagrams illustrating the use of restrictive policies. Check marks with brighter colours are evaluated as allowed actions.

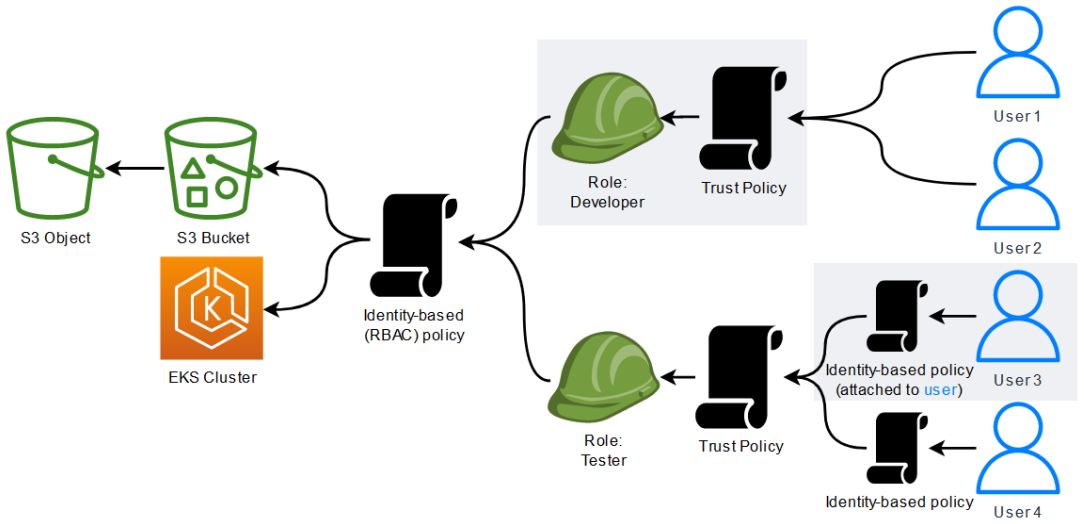

Figure 2.6: Illustration of how an RBAC model could be set up inside of AWS IAM.

resources), see the left side of Figure 2.4. The next policy type within the active policy group is the Resource-based policy. This policy is almost identical to the Inline policy, however, the IAM Entity and resource have changed positions (one resource, multiple users), see the right side of Figure 2.4.

Regarding the restrictive policies, Permission Boundaries are used to restrict Identity-based policies on the account level, see 'A.' in Figure 2.5. Similarly, SCPs are used to restrict IAM Entities at the organisation level. A Session Policy, however, is a bit different as it is passed as a parameter when you programmatically create a temporary session for a Role or *Federated User*; A Federated User is, for example, a user that has been imported from another authentication management system into the AWS system to be able to use one set of credentials for multiple workspaces. The passed Session Policy is then evaluated onto the entity that the temporary session was created for, see 'B.' in Figure 2.5. Furthermore, the Permission Boundary and Session Policy can be used in conjunction to create even more fine-grained access, see 'C.' in Figure 2.5. Note that Permission Boundaries do not restrict permissions granted by Resource-based policies that specify the IAM Entity as a Principal. Similarly, Session Policies also do not restrict permissions granted by Resource-based policies but have instead specified the Session ID as a Principal.

2.6.1 Role-based Access Control in Amazon Web Services Identity and Access Management

To implement RBAC in AWS, developers can utilise a feature called an IAM Role. An IAM Role can be seen as a group that some users are able to *assume*, unlike an actual User Group that users can be a part of, giving access to certain services or resources based on the attached policies. Therefore, to know which users are able to assume which roles, each role has one *Trust Policy*. This policy defines which Principals are allowed to assume the role, and under what conditions. Furthermore, a role is usually named something appropriate that would indicate its purpose. For example, a role could be named Developer, which would indicate that the role enables access to certain resources that developers need access to in order to perform their work. This relation is illustrated in Figure 2.6 where User 1 and User 2 are able to assume the Developer role, giving them access to two AWS resources, S3 bucket, which in turn stores S3 objects, and an EKS cluster.

However, whilst the Trust Policy defines who can access a role, they are not always directly *granting*, but only *allowing* some users to assume that role. This is similar to how

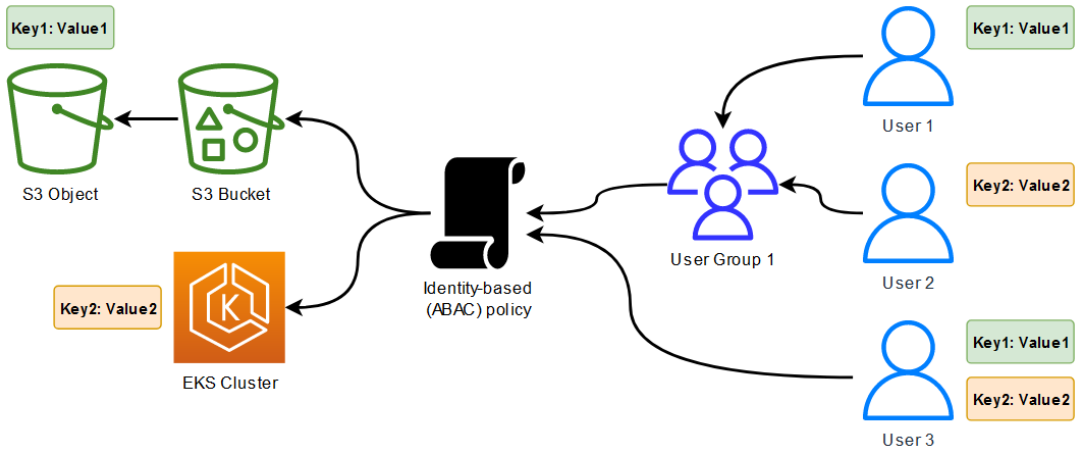

Figure 2.7: Illustration of how an ABAC model could be set up inside of AWS IAM.

Active- and Restrictive policies work for permissions, where Active policies actually *grant* some permissions whilst Restrictive policies only *allow* some permissions. Therefore, if the Trust Policy is written in a restrictive (allowing) manner, an Identity-based policy is needed to actually *grant* a user the permission to assume that specific role. This can be illustrated by Figure 2.6, where User 3 and User 4 have policies attached directly to them. This might be seen as a bit redundant, however, depending on the intended policy infrastructure and management of a system, it could come into good use.

2.6.2 Attribute-based Access Control in AWS IAM

For a developer to implement the ABAC model into AWS IAM, Tags have to be used. Tags are in simple terms *key-value pairs* attached to IAM Entities as metadata items, similar to the Version and Sid elements inside policies, that can be used in the Condition Block of a policy; see previous Figure 2.2 to understand the evaluation process of these key-value pairs within policies.

Moreover, as Tags are metadata items they can also be made to take any shape the developer would like them to take. For example, keys could be *Department*, *Team*, and *Project*, while the values could correspond to *Accounting*, *Team1*, *PoC1*; keys could also be used and compared directly, while leaving the value field blank, depending on developer preference. One thing to keep in mind, however, is that keys are case-sensitive in all cases except when used on IAM Users or Roles. If, for example, an IAM User has the key *Department* with value *Finance*, and another Tag is added with key *department* (lower case 'd') with value *HR*, then the former value will be replaced by the latter. Furthermore, make sure to never save any confidential or sensitive data into Tags, or any free-form text fields for that matter, as this can be openly seen when using for example the AWS Command Line Interface or AWS Software Development Kit[12].

To give an example of how Tags can be used, see Figure 2.7. In the figure, there are three users, and two of them are placed in a User Group. Furthermore, there is an attribute-written Identity-based policy attached to both the User Group and User 3, meaning that the policy grants access by matching Tags. This means that User 1, who has the *Key1: Value1* Tag, has permission to do some action on the S3 Object that has the same Tag. Furthermore, User 2 has the *Key2: Value2* Tag attached to itself, which is also attached to an EKS cluster (AWS Resource). This means that User 2 has permission to access the EKS cluster as well, whilst User 1 does not. Finally, User 3, who is not a part of any user group, has both the *Key1: Value1* and *Key2: Value2* Tags attached to itself. This means that User 3 can access both the S3 Bucket

[12]https://docs.aws.amazon.com/ARG/latest/userguide/security_data-protection.html

and EKS cluster. This final case can be especially good for individual permissions instead of using in-line policies.

3 Related Work

This chapter will present papers that are related to this thesis' work and write about how they have solved their particular problems. It will also go through papers relating to RBAC, ABAC, and similar access control models.

3.1 Related Work Pertaining to Role-based Access Control

This section will bring up work that is closely related to Role-based Access Control and also compare each work to this thesis.

3.1.1 Formal Definition, Feature Expansion, and Evaluation Specification

As previously stated in Background 2.5.4, Ferraiolo et al. [10] set out to create a formal definition of the RBAC model that is widely used today. Sandhu et al. [25] expanded upon this by identifying some well-known security principles that the RBAC model supported but did not enforce on its own. These principles were Least privilege, *Separation of duties*, and *Data abstraction*. Sandhu et al. specifically mention that "Theoretically, a system administrator could configure RBAC to violate these principels". The paper then moves on to categorise a base model (basic RBAC) and three feature models: the *Role Hierarchy Model* (built on the base model), the *Constraints Model* (built on the base model), and the *Consolidated Model* (built on the two previous models).

The Role Hierarchy Model adds role-within-role dependency, meaning a role can have both its own dependencies and inherit another role's dependencies. This can also be represented as an organisation's lines of authority and responsibility. The Constraints Model adds constraints for acceptability evaluations, meaning only accepted actions between entities can be carried out. Within this model, Sandhu et al. mainly describe three specific constraints; the first constraint is *Mutually Exclusive Roles*, meaning that a user can be assigned at most one role, in a role set, at one time; the second constraint is *Cardinality*, meaning that a role can at maximum only have a specified amount of concurrent users; the last constraint is *Prerequisite Roles*, meaning a user can be assigned role A only if they already have been assigned role B first. The last feature model is the Consolidated Model which merges the previous Role Hierarchy Model and Constraints Models into one model, to allow constraining of the role

hierarchy implementation directly; the Consolidated Model is also what would be expected of the RBAC model today.

Following Sandhu et al.'s different RBAC models, Ahn and Sandhu [2], in a later paper, try to create a "role-based authentication constraints specification" for evaluating constraints using logical *soundness* and *completeness* proofs. The specification presented is called RCL 2000, which is a generalisation of Sandhu and Ahn's previous version RSL99. RCL 2000 was made using a *formal language* approach instead of a *natural language* approach, which was used in RSL99. Their reasoning for this was that, even though natural languages have easier comprehension, they may be prone to ambiguities and do not support constraint generation and optimisation as the formal approach did.

While these three papers have defined, expanded, and evaluated the RBAC model as it is known today, some differences have to be noted in regard to AWS and this thesis. Firstly, the Consolidated Model's features, brought up by Sandhu et al., are already partially implemented in AWS. The only outlier is the Cardinality constraint within the Constraints Model which would not be possible to implement with normal policies. There would need to be some other metadata element attached to IAM Roles, which is currently not the case and not possible to implement due to the IAM architecture. Furthermore, in regards to Ahn and Sandhu's evaluation specification, AWS already uses its own evaluation method which cannot explicitly be managed or modified by a user or developer. There are, however, some best practices for IAM Policy creation that can be used to ease the evaluation process by, for example, utilising better grouping techniques for IAM Policy Statements and Resources.

3.1.2 Workflows and Tasks for Flexibility

As previously stated, the RBAC model has been widely defined and improved by many authors, however, some have found that it still lacks flexibility when it comes to some RBAC implementations. Leitner et al. [17] mention this issue in terms of *Workflow Systems* (WfS), meaning systems that manage repetitive processes and tasks that occur in some particular order. Leitner et al. explain that there are four aspects of change that need to be taken into account to accentuate flexibility. These aspects are *Control flow-*, *Data flow-*, *Administration-*, and *Service* changes. To test these aspects, Leitner et al. set four requirements that their proposed *Adaptive Workflow Role-based Access Control* (AW-RBAC) model must enforce for both flexibility and security to work.

The first requirement is completeness, meaning that only one access control model should be used for all aspects. If the aspects were to be divided into multiple access control models, it would increase the complexity of the access evaluation process and, in the worst case, make the access control weaker as some privileges might become negated due to a clash of access rights. The second requirement is *Support of user and context-dependent access rights*; user refers to an authorised person that can perform certain actions such as delete objects or remove dependencies; context means information such as location or time. The third requirement is *Maintainability of access rights*, meaning that it should be easy to add, modify, and delete access rights, as well as not attaching too specific context variables such as "process instance" or "process scheduling" to keep the RBAC model minimal (and maintainable). The last requirement is a Fine-grained definition of access control, meaning that there should be some possibility to define individual access rights when needed.

With the requirements defined, Leitner et al. used a use-case-based evaluation procedure by changing each aspect individually, on some test models that were created, and then implementing their *Adaptive Workflow Role-based Access Control* (AW-RBAC) model into a selected adaptive WfS. Although the implementation was still a work in progress, as AW-RBAC was only implemented on a subset of the WfS's components, their evaluation and conclusion state that the AW-RBAC model has been realised as a security service and can be implemented on any WfS.

Even though Leitner et al. wrote their paper about Workflow Systems back in 2011, Uddin et al. [32] show that the flexibility issue is still a lingering problem even in 2019. To handle this problem, the authors propose their *Authorising Workflow and Task-Role-Based Access Control* (AW-TRBAC) model. The difference with this model was that Uddin et al. seem to focus more on workflow used by developers, in comparison to Leitner et al.'s system workflow (WfS). Uddin et al. can with that create more specific requirements stating which roles are supposed to do what. An example of this would be the authors' requirements 1, 2 and 5 which respectively specify that: only the *Security Coordinator* can submit access requests, a *Process Owner* has to approve Security requests, and Service transfers should be approved by the Process Owner as well as revocation of existing, and provisioning of new, credentials. Another addition to Uddin et al.'s work is their use of *tasks*, meaning a fundamental unit of business work or activity. A task could also be called a 'Job function'. The paper moves on to extend the OASIS standard of the XACML policy language[1] to support the new requirements, and then implement this model using the Balana policy engine[2]. This was done to demonstrate that the model works on a real industrial use case from a financial institution.

Both Leitner et al. and Uddin et al. have seen the necessity of integrating flexibility into the current RBAC model and have both shown that there are multiple ways of doing so, mainly by integrating workflow access rights but also with task elements and well-defined requirement specifications when working with WfS. The difference in this thesis, however, comes down to one main point. Even though AWS provides a wide range of IAM Resources, which can be used for many different access control models, aspects such as workflow and task items do not have a direct connection to any IAM Resource. The workflow or task items would therefore have to be modified to fit or made into a more definite workflow environment, similar to how Uddin et al. did it, by putting restrictions on how access rights can be handled continuously instead of once during evaluation.

3.1.3 Attribute-inspired Role-based Access Control variations

Now that flexibility has been shown as an issue of the RBAC model, and some workflow solutions have been proposed, research has started to look into more heterogeneous solutions involving other models that take attributes and higher scalability, in comparison to RBAC, into account.

One such solution was proposed by Tang et al. [30], where they tried to combine the different role properties of RBAC with the broader and more attribute-based *Usage Control* (UCON) model. In comparison to the RBAC model, the UCON$_{ABC}$ model, as it is also called, integrates three core models: *Authorizations (A)*, *oBligations (B)*, and *Conditions (C)* [21]. Furthermore, the term "Usage Control" here is used to generalise the context of authorisations, obligations, conditions, continuity (ongoing controls), and mutability. Tang et al. [30], however, mainly use this model to gain access to the attachable attributes in order to create their own *Temporal, Spatial, Attributed, Workflowed Role-base Access Control* (TSAW-RBAC) model. Here, *Temporal* and *Spacial* means that one role can have different permissions when in different spatial locations and at different times while Attributed and Workflowed have similar meanings to earlier parts of this thesis.

Furthermore, Tang et al. propose ways to enforce the Temporal and Spatial attributes through *dynamic roles* and *dynamic policies* using, what they call, *Observer-* and *Supervisor Entities*; dynamic roles and policies mean that the calling entity is checked to be in the right position at the right time, namely spatially and temporally. If these conditions are not satisfied, the request will be denied. However, if the conditions are satisfied, and the calling entity leaves the spatial and temporal space, a revocation request will be sent by the Supervisor Entity and the calling entity will be suspended. The Observer Entity, on the other hand, is the

[1] https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=xacml
[2] https://github.com/wso2/balana

entity that initially assigns operations to objects and attaches attributes to those operations, according to predefined conditions.

Similarly, Qasim et al. [22] also mention the need for revocation strategies in their *Attributes Enhanced Role-based Access Control* (AERBAC) model. This model was built using normal RBAC with the addition of attributes from the standard ABAC model. The intention was to reduce complexity, by using ABAC, and simplify administration, by using RBAC, in larger organisations. However, while the AERBAC model does not bring any specific new implementations to the table, it does mention some interesting attributes that it used to simplify evaluation. For example, "otype" and "status" indicate which type of permission it is and what the status of that permission is (active/inactive), respectively.

The two papers by Tang et al. and Qasim et al., along with the UCON model, have further shown the need for flexibility in RBAC, however, they have also introduced the need for revocation strategies to enable continuity and mutability inside already established systems. However, while both Tang et al. and Qasim et al. talk about the need for attributes, neither of their models were actually implemented into a functioning system. The actual implementations were instead, as well as the revocation strategies, put into the authors' respective future work sections. While this might seem like a loss, AWS IAM has many features that could be applied and, to some degree, possibly function as revocation strategies. Therefore, these papers will help in finding these strategies.

3.2 Related Work Pertaining to Attribute-based Access Control

This section will bring up work that is closely related to Attribute-based Access Control and also compare each work to this thesis.

3.2.1 Attribute-based Access Control for Scalability

According to Biswas et al. [7], logical models can be conveniently used to perform access control because of two aspects: low cost and flexibility. Since there is no need to create roles with an upfront cost, like in RBAC, it is cheaper to use a logical model. Furthermore, there are no limits to how many attributes can be used in the access control computation, adding flexibility as well to a logical model. The authors mention that there is more than one way of expressing policies in ABAC variations, including *enumerated policies* which can be expressive. The authors go on to present an access control model, which is a variation of ABAC, named *Label-Based Access Control* (LaBAC) that uses labels to create these enumerated policies. The labels are grouped into *uLabel*s and *oLabel*s, which are connected to users and objects, respectively. The idea is that a user is assigned a single uLabel and an object is assigned a single oLabel. The labels consist of values that are derived from attributes, which in turn are derived from system and environment properties. The authors state that their model has a theoretical expressive power and flexibility since it can simulate other traditional models in it, including RBAC.

Internet of Things (IoT) is a concept where a growing number of physical objects, such as thermostats, smart homes, heating, and similar items, are connected to the internet and are thus able to communicate with each other [12]. Furthermore, these devices are able to share information and make decisions on their own. However, as soon as a device is connected to the internet, it needs proper protection. In the work by Sandhu et al. [6], the authors have expanded upon a previous access control model for AWS IoT (a service by AWS that manages IoT devices) by integrating ABAC capabilities into the model. Their motivation is that a flexible access control method is needed where there are a lot of parameters and environment variables. According to the authors, these scenarios can otherwise cause problems, such as role explosions, using today's industry-standard models, which is mostly the RBAC model. The authors present a model which they call ABAC AWS-IoTAC that expands upon the previous features by adding a fine-grained access control for IoT devices, data, and resources.

While the design of Sandhu et al.'s [6] proposed solution for AWS IoT would affect a *many-to-many* context, this thesis works in an access control system where the context instead is *many-to-one*. In IoT, many devices interact with many other devices, creating a cluster, whereas in AWS IAM, many users interact with one resource. Biswas et al.'s [7] work is somewhat similar to this thesis, in that they want to create a flexible model which would be able to include RBAC. However, they use enumerated policies while AWS IAM works with logic-based formulae, which is something this thesis revolves around as well.

3.2.2 Attribute-based Access Control in Cloud

The main design goal of Yu et al.'s [34] model is to create a secure model where the data owner has precise control over which user may access which data, while also obscuring those privileges as well as the data content itself from the cloud server. To achieve this goal, the authors have created a model that utilises and combines three different forms of encryption: *Key Policy Attribute-Based Encryption*, *Proxy Re-Encryption*, and *Lazy Re-Encryption*. This setup leads to one of the key features of this model, which is that the cloud computing servers are not able to read the plaintext information of any file on the server. Another key feature of this model is the scalability. This feature is achieved by the computational complexity that increases in one of two ways: proportionally or linearly. The first way depends on the number of system attributes, whereas the second way depends on the size of the structure detailing the users that have access, which the number of users in the system does not affect.

As stated previously, the RBAC model is an industry standard in access control but has limitations which can lead to role explosions, whereas the ABAC model does not have the same limitations. This is why Sandhu et al. [5] look at a way to integrate ABAC into OpenStack's access control framework *OpenStack Access Control* (OSAC). OpenStack is an open-source infrastructure as a service framework[3], with features similar to AWS, that also uses RBAC as a standard access control model. The authors propose a role-centric ABAC model for their work that allows them to utilise advantages from both the ABAC and RBAC models, and they call it the *User-Attribute Enhanced OSAC* model. Beyond the core components of the OSAC model, the authors have introduced user attributes which will help reduce the number of permissions determined by a user's role. In the end, by incorporating said benefits from the OSAC, the authors reach a model that enables fine-grained access control and avoids role-explosion, while also enabling users to be assigned the least possible permissions.

In Yu et al.'s [34] work, their context regards a single user who wants to store data on the servers, which belong to the user itself, and should be able to delegate access rights to other users. The data is also obfuscated and not readable by the server itself. These features are similar to how companies generally prefer their access control today, to be able to delegate access rights to other users in a fine-grained manner. The difference lies in ownership and delegation of the data, where the company itself, a collection of users, owns the data and has default access from an admin account, whereas in the context of the related work, a single user owns all the data tied to them. The company needs to decide internally who gets access to what, and who gets admin privileges, etc., whereas the user who owns all the data has all access rights by default, and can delegate privileges as they wish. In the work by Sandhu et al. [5], similarly to this thesis, they try to implement a solution to a cloud computing system that runs RBAC as access control and try to find a solution integrated with ABAC. However, while they use RBAC as a base and then add attributes to it, this thesis aims to use ABAC as a base and then add roles to it for management capabilities.

[3] https://www.redhat.com/en/topics/openstack

3.3 General Related Work

Eiers et al. [8] state that complex policy specifications are harder to manage and are more prone to unauthorised access to data. Without the necessary tools, it becomes increasingly difficult to protect private information due to the increasing size of systems and their general complexity. Therefore, the authors present a method that compares different policies' permissiveness to each other and then quantifies this using a model counting constraint solver. The authors also present a heuristic to transform policies to improve the model counting performance.

While Eiers et al. use their counting constraint solver to evaluate policies' permissiveness, this thesis uses human input which is gathered through a survey. A computer can only give answers based on input, or certain facts, whereas humans tend to be biased and act on emotion rather than rational thoughts when asked about how well a certain system performs.

4 Method

This chapter will present the methods used in order to produce the results. Furthermore, this chapter will describe the survey used, which steps were taken to construct the proposed access control mode, as well as how the experimentation was conducted.

4.1 Survey Method

A survey can be conducted for several purposes, and Wohlin et al. [33] best defines when you should conduct a survey with the following scenario: "Surveys are conducted when the use of a technique or tool already has taken place or before it is introduced." A survey can be used to capture the current status of a situation, and in this thesis's case, a workflow environment. The employees at NIRA Dynamics who work with AWS IAM had worked with the current system before this thesis started, and thus a survey in this case aimed to identify, among others, the following aspects such as what worked, what did not work, and what could be improved. The information gathered from the survey was used to create a baseline and understanding of how well it was to work in AWS IAM using the current model and settings at the time. In other words, an explanatory survey was conducted in order to gather insight regarding the use of AWS IAM at the time, and what could be improved.

The survey was conducted through interviews with the participants. Google Forms[1] were used to mediate the questions to the participants, as well as store the survey answers, because of its design and overall usefulness in creating and using the survey. All answers were kept anonymous and did not contain any sensitive or classified information. The chosen participants were employees working closest with AWS IAM, in certain teams, and the aim was to interview as many of them as possible during the survey. The survey type was a qualitative one, not quantitative, but interviewing more people meant identifying as many aspects as possible by having the interviewees recount their experience working with AWS IAM.

In total, 9 people were interviewed during the survey. The reason for doing interviews, as opposed to just sending the participants the online form, was to be able to ask follow-up questions and to elaborate on different topics that would have been difficult to elicit from the participants otherwise.

[1] https://docs.google.com/forms/

The tables in Appendix A present all the questions that were present in the survey, grouped by their respective category. The questions that have a linear scale as an answer type have two different notations. The ones referring to how much time something takes have answers ranging from *Very Short* to *Very Long*, while the ones referring to how often an activity occurs will have the options *Never*, *Rarely*, *Sometimes*, *Often*, and *Very Often*. Since not every question was mandatory, each row in the tables will show either M or O, which will indicate if the question was mandatory or optional, respectively.

4.2 Proposed Access Control Model

To develop an access control model that contained the requirements and specifications needed of the proposed access control model, the designing process was separated into three different iterations: Attributes, Roles, and Access Management. These iterations helped distinguish what each added feature contributed to the proposed access control model. Furthermore, by separating each feature into iterations, it was easier to verify that each feature could be implemented in AWS and cooperate with each other. Moreover, this helped to be able to clearly motivate each added feature and why it was needed for the proposed access control model.

Each feature from the different iterations was implemented in AWS using the S3 resource due to its popularity. Furthermore, the company that this thesis worked with used S3, among many other resources, which made it a relevant and useful tool to have experimented with and developed on.

The first version of the design included only attributes, which is the core feature of the ABAC model [14]. Since flexibility and scalability were some of the properties that the proposed access control model needed, it was important to implement ABAC correctly, because that model would act as a foundation for all additional features. However, since S3 did not fully support the ABAC model[2], the model that was implemented in this iteration was not a complete representation of the ABAC model's capabilities. For example, it was only possible to attach attributes (tags) on the objects themselves, not the buckets or folders that the objects were placed in. This made it difficult to apply some form of group permission for buckets, and folders, using tags. This did, however, enable fine-grained access which was, according to Leitner et al. [17], a sought-after property of using the ABAC model for access control. With this in mind, four different solutions were created; a solution in this case means a policy, or policies, that **can** be used for access control, however, it might not be the most appropriate for the final proposed access control model.

[2]https://docs.aws.amazon.com/IAM/latest/UserGuide/reference_aws-services-that-work-with-iam.html#storage_svcs

Policy 4.3 Identity-based policy for S3 resources (Public Read(List))

```
1  {
2    "Version": "2012-10-17",
3    "Statement": [
4      {
5        "Sid": "RequiredToSeeBucketsAtAll",
6        "Effect": "Allow",
7        "Action": [
8          "s3:ListAllMyBuckets",
9          "s3:GetBucketLocation",
10         "s3:ListBucket"
11       ],
12       "Resource": "*"
13     },
14     {
15       "Sid": "AllowsUsersWithCorrectTagToAccessObjectsWithinAllBuckets",
16       "Effect": "Allow",
17       "Action": "s3:GetObject",
18       "Resource": "arn:aws:s3:::*/*",
19       "Condition": {
20         "StringEquals": {
21           "s3:ExistingObjectTag/tag-key": "${aws:PrincipalTag/tag-key}"
22         }
23       }
24     }
25   ]
26 }
```

4.2.1 Iteration 1: Attributes

The first solution, see Policy 4.3, did not focus on the grouping permissions as explained previously but only on who could actually read or download the objects within buckets or folders. This meant that the solution used a *Public Read(List)* principal, meaning that all users could see the object's names within buckets and folders, but not its contents. In this Identity-based policy, the actions present in the first statement on code lines 8-10 are always necessary for the user to be able to do any other action on S3 Buckets. One such example would be the *s3:GetObject* action in the second statement, however, almost every other action that involves S3 buckets will need these three actions to work as well. Therefore, almost every policy moving forwards will have these three actions present. To make sure that the second statement did not allow all users to read or download objects, however, a condition block was also added to make sure that the principal's tag (in this case a user's tag) matched that of the existing object's tag, that the user is trying to download. Finally, the policy was attached to a user group that all current users were a part of. The policy could also be attached to the users individually, however, that would only create duplicate policies with no further functionality than what a user group could bring.

The second solution, see Policy 4.4 below, however, was created with more security in mind as most companies might not want to give all users Public Read(List) capabilities. This solution worked by extracting the *s3:ListBucket* action from the first statement and inserting it into its own statement. This statement made sure that users with a specified tag could list all buckets, however, all other users without this specific tag would not be able to do any further actions within S3. The reason for this was that now all users without the specified tag would be excluded from one of the three prerequisite actions that are needed to be able to use for example the *s3:GetObject* action. In conclusion, this solution puts users into two groups, one group that has access to list and use all buckets, and one group that does not have access to list or use any buckets at all. Similarly to the first solution, this solution was also attached to a user group that all users were a part of.

Policy 4.4 Identity-based policy for S3 resources (only users with correct tag)

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "RequiredToSeeBucketsAtAll",
      "Effect": "Allow",
      "Action": [
        "s3:ListAllMyBuckets",
        "s3:GetBucketLocation"
      ],
      "Resource": "*"
    },
    {
      "Sid": "AllowsUsersWithCorrectTagToListAllBuckets",
      "Effect": "Allow",
      "Action": "s3:ListBucket",
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "aws:PrincipalTag/tag-key": "tag-value"
        }
      }
    },
    {
      "Sid": "AllowsUsersWithCorrectTagToAccessObjectsWithinAllBuckets",
      "Effect": "Allow",
      "Action": "s3:GetObject",
      "Resource": "arn:aws:s3:::*/*",
      "Condition": {
        "StringEquals": {
          "s3:ExistingObjectTag/tag-key": "${aws:PrincipalTag/tag-key}"
        }
      }
    }
  ]
}
```

Policy 4.5 Identity-based policy for S3 resources (using a naming standard)

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "RequiredToSeeBucketsAtAll",
      "Effect": "Allow",
      "Action": [
        "s3:ListAllMyBuckets",
        "s3:GetBucketLocation"
      ],
      "Resource": "*"
    },
    {
      "Sid": "AllowsUsersWithCorrectTagToListBuckets",
      "Effect": "Allow",
      "Action": "s3:ListBucket",
      "Resource": "arn:aws:s3:::project:*",
      "Condition": {
        "StringEquals": {
          "aws:PrincipalTag/tag-key": "tag-value"
        }
      }
    },
    {
      "Sid": "AllowsUsersWithCorrectTagToAccessObjectsWithSameTagWithinBuckets",
      "Effect": "Allow",
      "Action": "s3:GetObject",
      "Resource": "arn:aws:s3:::project:*/*",
      "Condition": {
        "StringEquals": {
          "s3:ExistingObjectTag/tag-key": "${aws:PrincipalTag/tag-key}"
        }
      }
    }
  ]
}
```

To try and extend these grouping permissions further, a third solution was created, see Policy 4.5. In this solution, a naming standard was proposed to be able to give different buckets different access tags depending on who should be able to use them. The naming standard is illustrated in the second, and third, statement's resource parameter as *project:**. This means that the specified resource has to start with a specific project name, or id, followed by a colon and finally a bucket name of choice (*, "wildcard"). This makes it possible to group all buckets with the same project name into one statement whilst also checking so that a user has the correct tag, in this case preferably the project name. In conclusion, this allows buckets to be grouped into as many groups as tags used, therefore gaining a bit more flexibility and security between buckets. Similarly to the first- and second solution, this solution was also attached to a user group that all users were a part of.

Policy 4.6 Identity-/Resource-based policy for S3 resources (no naming standard)

```
-------------- Identity-based policy --------------
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "RequiredToSeeBucketsAtAll",
      "Effect": "Allow",
      "Action": [
        "s3:ListAllMyBuckets",
        "s3:GetBucketLocation"
      ],
      "Resource": "*"
    }
  ]
}

-------------- Resource-based policy --------------
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowsUsersWithCorrectTagToListAllBuckets",
      "Effect": "Allow",
      "Principal": "*",
      "Action": "s3:ListBucket",
      "Resource": "<attached-resource>",
      "Condition": {
        "StringEquals": {
          "aws:PrincipalTag/tag-key": "tag-value"
        }
      }
    },
    {
      "Sid": "AllowsUsersWithCorrectTagToAccessObjectsWithinAllBuckets",
      "Effect": "Allow",
      "Principal": "*",
      "Action": "s3:GetObject",
      "Resource": "<attached-resource>/*",
      "Condition": {
        "StringEquals": {
          "s3:ExistingObjectTag/tag-key": "${aws:PrincipalTag/tag-key}"
        }
      }
    }
  ]
}
```

While the previous solution solved the problem of being able to group buckets into more than two groups, it also had to implement a naming standard to work. However, when developing an environment, naming standards can quickly increase the complexity due to the need of remembering how everything is supposed to be structured. To try and avoid this issue the fourth, and final, solution was created, see Policy 4.6. By separating the previous Identity-based policy into one Identity-based and one Resource-based policy, the naming standard could be omitted, since Resource-based policies are bucket specific, meaning that each bucket has its own usable policy for allowing certain actions within that specific bucket.

This removed the need for specifying which buckets were to be involved in which groups as each bucket now asked for the same condition, namely that the user had the correct tag when trying to perform an action. However, whilst the Resource-based policy could be used instead of the naming standard, the Identity-based policy was still needed due to the actions *s3:ListAllMyBuckets* and *s3:GetBucketLocation* which needed to be placed on an IAM Entity (in this case a user or user group).

4.2.2 Iteration 2: Roles

Every solution has its limitations, and while the ABAC model is flexible and scalable, it is not as suited for administrative purposes as RBAC [15]. However, since the proposed access control model was in part going to solve a problem in an industrial environment, it needed administration features to be able to enforce some form of admin privileges, collective privileges, etc. Therefore, the proposed access control model would use both attributes and roles to gain flexibility, scalability, and administrative abilities. The idea behind integrating roles into an ABAC model was to utilise the advantages of RBAC to eliminate the weaknesses of ABAC. One great example of this is that the ABAC model does not work across multiple accounts in and of itself. This is where RBAC can help as roles benefit from being Federated Users, meaning the users from one AWS account can transcend to another by assuming a role, which also helps in an industrial environment. Another reason for integrating roles into the model was the idea of attaching one policy to multiple users at the same time. Previously, this was done using a user group, however, as these cannot be tagged it would not work in a cross-account environment. Roles, however, have the additional support of tags which would be needed for this type of model to work correctly.

With these features in mind, the IAM Role feature was used to create a role hierarchy using the roles *analyst*, *developer*, and *admin*, whom each had an increasing amount of privileges compared to the other; different role's actions must be a proper subset[3] of each other. An example of this would be that the analyst only had read access, the developer role had read, write, and update access, whilst the admin had read, write, update and delete access.

Before integrating roles into Iteration 1, however, users must first be able to assume a role in a simpler manner than having to specify each user within each Trust Policy, creating large policies that will be difficult to manage. To solve this, attributes were implemented into the IAM Role's Trust Policy. This meant that all users within any account in the organisation and with a certain tag called *role*, could be matched with the IAM Roles that had the same tag key-value pair attached to themselves, see Policy 4.7 below. An example of this would be that both a user and a role have the tag (key-value pair) <role>:<developer> which would result in the user being able to assume the role of developer.

After a user has been able to assume the correct IAM Role, according to their matching tags, the user should also get some further permissions according to the assumed IAM Role. In this case, two solutions were created. The first solution policy was attached directly to the role, whilst the other one was attached to the role and resource separately, similar to the last two solutions of Iteration 1. For simplicity, the following solutions were also written using actions named *s3:Read*, *s3:Write*, *s3:Update*, and *s3:Delete* for S3; the prerequisite actions *s3:ListAllMybuckets*, *s3:GetBucketLocation*, and *s3:ListBucket*, that were brought up in Iteration 1, are aggregated under the *s3:Read* action in this case. However, whilst the *s3:Read*, *s3:Write*, *s3:Update*, and *s3:Delete* actions were not actual actions in S3, they will be used to simulate the previously explained access hierarchy. In this case, the analyst role would only have access to the *s3:Read* action, the developer role would have access to the *s3:Read*, *s3:Write*, and *s3:Update* actions, whilst the admin role would have access to all of the actions.

[3] https://mathinsight.org/definition/proper_subset

Policy 4.7 Trust Policy for an IAM Role

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllUsersWithinAccountWithSameTagAsRoleCanAssumeRole",
      "Effect": "Allow",
      "Principal": {
        "AWS": "*"
      },
      "Action": "sts:AssumeRole",
      "Condition": {
        "StringEquals": {
          "aws:PrincipalTag/role": "${aws:ResourceTag/role}"
        }
      }
    }
  ]
}
```

Policy 4.8 Identity-based policy for a Developer Role (Resource Specific)

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowsReadWriteUpdateDeleteForResourcesWithSameTagAsRole",
      "Effect": "Allow",
      "Action": [
        "s3:Read",
        "s3:Write",
        "s3:Update",
        "s3:Delete"
      ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "aws:PrincipalTag/role": "${aws:ResourceTag/role}"
        }
      }
    }
  ]
}
```

As explained, the first solution focused on attaching the policy to the role itself. Therefore, to extend the previous example, where a user assumed the developer role through ABAC, the user is now granted actions according to the Identity-based policy that is attached to the Developer role, if-and-only-if the S3 resources' tags match that of the assumed role as well. For a developer, these actions might be those shown in Policy 4.8 except the *s3:Delete* action. Furthermore, this policy was also created for only one type of resource, namely S3. If there were to be more AWS resources, for example, EC2 and EKS, these would have their own separate policies with similar statements within. On the other hand, the policies could also be combined into one single policy, however, if many more resources were to be added it would become confined and difficult to manage, see Appendix Policy E.13 for an example of this.

Policy 4.9 Identity-/Resource-based policy for all three Roles

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowsReadWriteUpdateDeleteForResourcesWithSameTagAsRole",
      "Effect": "Allow",
      "Action": [
        "s3:Read"
      ],
      "Resource": "<attached-resource>",
      "Condition": {
        "StringEquals": {
          "aws:PrincipalTag/role": "analyst"
        }
      }
    },
    {
      "Sid": "AllowsReadWriteUpdateDeleteForResourcesWithSameTagAsRole",
      "Effect": "Allow",
      "Action": [
        "s3:Read",
        "s3:Write",
        "s3:Update"
      ],
      "Resource": "<attached-resource>",
      "Condition": {
        "StringEquals": {
          "aws:PrincipalTag/role": "developer"
        }
      }
    },
    {
      "Sid": "AllowsReadWriteUpdateDeleteForResourcesWithSameTagAsRole",
      "Effect": "Allow",
      "Action": [
        "s3:Read",
        "s3:Write",
        "s3:Update",
        "s3:Delete"
      ],
      "Resource": "<attached-resource>",
      "Condition": {
        "StringEquals": {
          "aws:PrincipalTag/role": "admin"
        }
      }
    }
  ]
}
```

The second solution was to use the S3 Resource-based policy that is attached to each bucket individually. However, as there is only one of these Resource-based policies for each bucket, the solution became inverted and each statement inside the policy instead had to check so that some actions were only allowed if a specific role was the one accessing the bucket, see Policy 4.9. Note that this policy is quite large, and in the case of more roles, this policy would possibly grow even more. Furthermore, the prerequisite actions within the *s3:Read* action would still have to be put on the roles themselves, as seen in Iteration 1 Policy 4.6, as they cannot be placed on the bucket directly.

4.2.3 Iteration 3: Access Management

As Iterations 1 and 2 primarily focused on how a combination of ABAC and RBAC could work, little was said about how this combination is supposed to be set up or who should manage it moving forwards. This issue was shortly addressed by Leitner et al. [17], and more extensively addressed by Uddin et al. [32], as shown in Related Work 3.1.2, where they set up requirements specifying how the access rights were supposed to be handled, and by whom. To put these requirements into perspective, if for example an existing role can create a new role and put tags on elements without restrictions, that same role could create a new

Policy 4.10 Identity-based policy for Ensuring No-Read-Up and No-Update-Up

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:GetObject"
      ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "s3:ExistingObjectTag/role": [
            "developer",
            "analyst"
          ]
        }
      }
    },
    {
      "Effect": "Allow",
      "Action": [
        "s3:PutObjectTagging"
      ],
      "Resource": "*",
      "Condition": {
        "StringNotLike": {
          "s3:ExistingObjectTag/role": "admin"
        }
      }
    }
  ]
}
```

admin role with even more permissions than they should be allowed to have, leading to security problems. Similarly, if we did not have tags, some roles need to act as a middle hand and be able to change policies themselves. This could, however, lead to a security issue as policies handle all of the access rights within AWS IAM.

Therefore, to properly manage access control moving forward, some security measures have to be implemented. In Uddin et al.'s case, the solution was to have dedicated roles such as Security Coordinator and Process Owner. The Security Coordinator was the one that submitted access requests, while the Process Owner was the one allowing these accesses. In larger systems, however, this could constrain the workflow a lot depending on how many levels of Process Owners there are. For example, if the Process Owner can grant full access rights and some other roles below can grant a subset of those access rights then the Process Owner can delegate some of the work to other people, instead of controlling everything themselves. If this is not the case, however, and the Process Owner has to allow all access rights themselves, there is a high risk of a bottleneck happening when multiple Security Coordinators want different access rights at a rapid pace. This could further increase the possibility that the Process Owner takes shortcuts, by giving the Security Coordinators too much access, which would deteriorate the overall security of the system. Therefore, by following the previous examples, two solutions that could increase the manageability of the proposed access control model were created.

The first solution was to outsource the responsibility of setting up access rights to each role within the role hierarchy, which was described in Iteration 2. For example, if a role had the access rights of a developer, they could only read objects which had either the same tag or any tag below their own, to enforce a no-read-up principle. Furthermore, they could only change the tags on objects that already either had their own tag, or any tag below their own, see Policy 4.10. This made sure that new resources were reachable by the same people that created them, as well as restricting people without the proper tags to modify the access level of certain objects. This solution did, however, allow for developers to also add tags higher than their own, with the reason that a singular developer should be able to help an

Policy 4.11 Identity-based policy for a New AccessManager Role

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "iam:*"
      ],
      "Resource": "*"
    },
    {
      "Sid": "SomeResourceSpecificActionsLikeTagging
      ...
    }
  ]
}
```

administrator to create an object that the developer would then lose control over. However, while this might work for some cases, sometimes it might be better to restrict this kind of behaviour and instead go for an already proven integrity model, namely the Biba model, as brought up in Background 2.5.3. Note that, for simplicity, this solution was presented without the other S3 actions, and even though they could be put into another policy, they have not been omitted in the tested code.

The second solution, for managing access control, was to create a separate role called *AccessManager*. This role was responsible for setting up all IAM elements and access rights between users, roles, and resources, in comparison to the role hierarchy, whose main purpose was more focused on reading, creating, and deleting resources. In conclusion, this role would need all of the actions granted by the IAM action library, namely *iam:**, however, it would also need some resource-specific actions, similar to the previous solution's tagging procedure, see Policy 4.11.

4.3 Experimentation Method

The proposed access control mode will henceforth be referred to as Entity-centric Attribute- & Role-based Access Control (EARBAC), and the justification for this name can be read later on in the Results chapter. To validate EARBAC and test its performance, an experiment was conducted. The experiment consisted of a set of different tasks that a set of subjects performed in a constructed environment in AWS IAM. However, after the initial experimentation was performed, it was noticed that the results were biased towards the RBAC model since it had avoided steps that would have been prominent in a realistic scenario. Therefore, these tasks were retroactively performed and measured in order to more accurately represent a true RBAC model. According to Wohlin et al. [33], there are five important steps to follow when carrying out a controlled and valid experimentation, which involve: *Scoping*, *Planning*, *Operation*, *Analysis and interpretation*, and *Presentation and package*. Therefore, this thesis will focus on the first three of these, as the last two steps are represented by the thesis itself.

4.3.1 Scoping

By defining the goals of this experimentation, the scope can be clearly defined. This is done by a goal definition template in combination with a final goal summary.

Goal Definition

In the goal definition template, there are five questions that have to be answered: *Object of Study*, *Purpose*, *Quality Focus*, *Perspective*, and *Context*.

Object of Study. The object of study was the participants at NIRA Dynamics and their ability in terms of performance based on the access control model and their own previous experiences using AWS IAM.

Purpose. The purpose of the experiment was to evaluate the individual efficiency based on which access control model, either RBAC or ABAC, was used when managing (adding, deleting, modifying) AWS resources and users. The experiment provided insight into which model generally had higher learnability and ease of use in AWS.

Quality Focus. The main effect studied in the experiment was the individual performance using different access control models to manage permissions in AWS IAM. Here, one specific aspect is emphasised, namely *Efficiency* (Time/Task). Another effect studied was how easy it was for an individual to implement the access control model presented to them. This was done by counting the number of lines of code after the final model was implemented, this measure was called *Complexity*. The final effect studied was how many times a policy had to be changed after the final model had been set up, this measure was called *Risk*.

Perspective. The perspective on this experimentation was from the point of view of the researchers and the general AWS IAM user or administrator. This means that the researchers would like to know if there were any systematic differences presented in the individual performances based on previous experience and the use of different access control models. Furthermore, this would be applied to the general AWS user or administrator when setting up their own access control.

Context. The experiment was run within the context of access control within AWS IAM. Moreover, this thesis was held in cooperation with NIRA Dynamics. The experimental context characterisation was *multi-test within object* where the subjects were chosen among the employees at NIRA Dynamics and the object was an AWS IAM environment. A set of tasks were performed by the subjects in order to measure the performance of EARBAC. As the subjects had mostly different previous experiences, as well as were not fully integrated in either the RBAC or the ABAC model, this meant that the experimentation could be judged as being controlled, since the subjects were picked at random during the experimentation.

4.3.2 Planning

After the foundation has been laid, the planning phase prepares for how the experiment is to be conducted. This phase is divided into seven steps: *Context Selection*, *Hypothesis Formulation*, *Variable Selection*, *Selection of Subjects*, *Experiment Design Type*, *Instrumentation*, and *Validity Evaluation*.

Context Selection

The context of the experiment was the IAM system in AWS. The experiment was held in an industrial system but not in an industrial environment, meaning that the access control system was run in a simulated environment that matched the real production environment. Hence, the experiment is regarded as off-line. The subjects in the experiment were NIRA employees, meaning that the experiment was done in collaboration with professional users. Furthermore, since this thesis focuses on a pre-existing problem of access control, which is shown by NIRA and is prevalent in previous works, this experiment is regarded as a real problem. Finally, the context of the experiment will be specific to access control in AWS, not general access control of other systems, such as Microsoft Azure, Google Cloud, etc.

Hypothesis Formulation

By taking the goal definition from the last section, a hypothesis formulation could now be realised as:

> Through the background, related work, and survey presented in this paper, both the papers and NIRA staff have highlighted the need for greater flexibility and scalability in access control. With this information, it would be appropriate to say that EARBAC, as compared to the RBAC model, supports these needs for flexibility and scalability, which would in turn increase staff efficiency, regardless of prior experience. In addition, it would also be appropriate to say that EARBAC has a higher setup complexity compared to the RBAC model. However, after setup, a reduction in policy restructuring and overall risk may be observed.
>
> Null Hypothesis:
> There is no difference in Efficiency between the different access control models. H_0 : Effi(EARBAC) = Effi(RBAC). There is also no difference in Complexity (lines of code) between the two models. H_1 : Comp(EARBAC) = Comp(RBAC). Finally, there is no observed difference in Risk (changed policies after final implementation) between the two models. H_2 : Risk(EARBAC) = Risk(RBAC).
>
> Alternative Hypothesis:
> H_3: Effi(EARBAC) $\neq$ Effi(RBAC),
> H_4: Comp(EARBAC) $\neq$ Comp(RBAC),
> H_5: Risk(EARBAC) $\neq$ Risk(RBAC)
>
> Measurements needed: Access control models (EARBAC or RBAC), Efficiency (Time/-Task), Complexity (Lines of Code), and Risk (Policy Changes).

From these hypotheses, the following data was collected:

- Access control model: measured by EARBAC or RBAC (nominal scale).
- Efficiency: measured as Time/Action. Action means performing a certain task in AWS, and a specific set of tasks will be shown later in this chapter. This means that the time needed to perform each action will be measured and compared individually, as well as in combination, between different participants and access control models.
  Time is measured in minutes (ratio scale).
  Actions are labelled by certain tasks (nominal scale).
- Complexity: measured as Lines of Code (LoC). LoC refers to the lines of code in each policy in the access control system.
- Risk: measured as the number of policy changes needed, to update access rights, after setup.

Variables Selection

The independent variable here was the access control model (controlled). Another independent variable considered was the experience of personnel, however, since this was not something that could be seen as a controlled- or fixed variable, due to a small subject pool, it was not used. However, through instrumentation, some attempts were made to even the experiences of the subjects so that the experiment was carried out in a more controlled manner. The dependent variables were Efficiency, Complexity, and Risk.

Task Nr.	Task Description	Expected Access Result
1	Setup Assume Role for User	Allowed
2	Create Identity-based policy for Roles	Allowed
3	Developer writes Object new_object.txt	Allowed
4	Developer updates Object new_object.txt	Allowed for EARBAC
5	Admin updates Object new_object.txt	Allowed

Table 4.1: Table showing a set of tasks each subject will perform during experimentation

Selection of Subjects

The subjects (NIRA personnel) were chosen using *simple random sampling*, meaning that they were placed in a list and chosen at random.

Experiment Design

As per the experiment definition, there are four aspects that have to be considered when choosing an experimentation design. These aspects are *Randomisation*, *Blocking*, *Balancing*, and *Standard design types*.

Randomisation. The object was not assigned randomly to the subjects as all participants would apply some treatment to the same base environment (object). On the other hand, subjects were selected at random from the population. Moreover, the model that the subject started testing was also selected at random during the experimentation. This was to limit the amount of bias towards any specific model as they were measured and compared against each other.

Blocking. No systematic approach to blocking was applied. The reason for this was to get a better generalisation for other AWS IAM users with likewise different prior experiences.

Balancing. For each subject, the treatment, either RBAC or ABAC, was chosen at random, however, it was also balanced so that each model had the same amount of subjects for better test accuracy.

Standard design types. As per the definition, hypothesis, and measurements stated previously, the design became *one factor with two treatments* with *paired comparison design*[33]. The factor was the access control and the treatments were either RBAC or ABAC. The dependent variables were measured on a ratio scale and were thereby suited for a parametric test. In this case, the *Paired t-test* was used and it was also used to evaluate the hypotheses, and for further information on how the test works, see the paper by Wohlin et al. [33].

Instrumentation

To ensure that the experiment is carried out correctly and successfully, the following instruments are needed: *Experiment objects*, *Guidelines*, and *Measurements*.

Experiment objects. As previously stated, the object of this experiment was an environment within AWS IAM that was to be tested and modified by the subjects through the access control factor. This environment was created with the prerequisite 3 Users, 4 Resources, 3 Roles, 2 Policies, and 3 Tags. This environment was modified by each subject during the experimentation.

Guidelines. Guidelines were created for the subjects to follow during each task of the experimentation. Each task has been listed in Table 4.1 along with its expected result. For Task 4, the results differed because EARBAC could perform an action that RBAC could not. The guidelines, which are presented in Appendix B, acted as instructions for the subjects and helped them understand how to perform each task. Furthermore, each task was timed.

Measurements. Each task that a subject performed was timed in order to evaluate the differences in Efficiency between RBAC and EARBAC. Additionally, after each task, the subject was asked to provide feedback on whether they thought one model was easier to use than

the other. Since each of the subjects did not start the experiment with the same model, they were able to provide qualitative feedback on the perceived usability of the different models. Furthermore, the number of policies and total lines of code were measured once in order to evaluate the differences in Complexity between the two models. The reason why the Complexity metric only needed to be measured once was that the code in the environment did not change between the tasks or between the subjects. Finally, Risk was measured by counting the number of times a policy had to be changed, after either model had been implemented, to increase access rights. In this case, access rights meant a user gained new access to assume a role or to be able to download a new object. Risk was also measured once, for the same reasons as for Complexity.

Finally, the steps that were avoided in the RBAC model were measured as well, after the initial experiment. These steps included fetching the ARNs of each user and resource involved in the tasks that the subject performed. Furthermore, these steps were performed and measured using two other subjects two times each, meaning four times in total. Using these four measurements, an average was calculated and added to each subject's corresponding task time. This way, a new final result was calculated, from the new task times, that better represented the Efficiency metric of the RBAC model.

Validity Evaluation

As this thesis was heavily focused on qualitative research and has built its foundation in applied research, there are some priorities that had to be followed when evaluating the validity of this experimentation. This order was from highest priority to lowest: *Internal-*, *External-*, *Construct-* and *Conclusion Validity*.

Internal Validity. The experimentation was conducted with ten subjects, which could have resulted in potential outliers. Since there were only a certain number of people available to experiment on from one company, these outliers were difficult if not impossible to avoid. The subjects might have had different familiarities with the AWS console, in which the experiment was conducted, which might have caused outliers in this area as well. However, all the subjects had a guide and explanation of the environment and which actions were available to them at all times. These aids hopefully combated this problem and evened out the different experiences of the subjects. Since the experiment was based on the subject performing the tasks using one model before the other, they could have developed a bias towards a certain model or gained knowledge of the AWS console which could have aided them in the second phase. To combat this bias, half of the subjects started with the RBAC model, while the other half started with the EARBAC model.

External Validity. By only conducting the experimentation at NIRA Dynamics, and no other company, the results put forth could be interpreted as specific and not generalisable. However, while only experimenting on one company, the subjects themselves held varying experiences as well as different positions within the company, which could be seen as a more generalised environment that would fit many other companies as well. Due to time constraints on the overall thesis, only a part of all subjects were experimented on. This could create some misrepresentation of the company itself, however, as all subjects were picked at random, within the company, the results were as generalised as possible.

The experimentation was done in a testing environment constructed in the AWS console, which is a graphical user interface on the web. Since the testing environment in the AWS console consisted of a number of specific actions selected to be able to perform certain tasks while other possible features of the console were showing, it is possible that the subjects might have been confused or distracted by this. Furthermore, this could have made it difficult for the subjects to navigate the console if they were unfamiliar with certain features in the graphical user interface. The guidelines were introduced to combat this problem, and they informed the user what actions were needed to accomplish the tasks given. Each available action, in the context of the experiment, was pointed out to the subjects and they were given

as much aid as possible to help distinguish between which features were necessary for the experiment and which were not.

Construct Validity. There are some threats to the construct validity of the experiment, which include the results and theory of the experiment. One such threat is the *mono-method bias* that can occur if there is only one type of measurement which would result in a measurement bias and the results of the experiment could then be misleading. While only the Efficiency metric was measured for each experimentation session, the Complexity- and Risk metrics were measured as well, but only one time during the experimentation because their measurements were static throughout the experiment. Another threat is *confounding constructs and levels of constructs* which means that the existence of a construct such as prior experience with a certain tool is confounded with the level of said experience. In this experimentation's case, each subject's experience with AWS was treated as an independent variable and thus has to be taken into consideration. What was not taken into consideration, was how much previous experience each test subject had, whether it was just a month or several years. However, the guidelines that were put in place helped combat this issue as well since they gave detailed instructions for what the subjects should do for each task.

Interaction of testing and treatment is also a viable threat, and it describes the issue where the subjects are aware of the purpose of the experiment and thus try to improve their results by avoiding common mistakes, for example. The subjects in this experimentation were informed that a timer was used to measure their performance, and that could have affected their performance or made them stressed, despite reassurance that they did not have to feel any pressure to perform at their best. Furthermore, before each subject began experimenting, they were told the purpose of the experiment so that they would not have to guess. This relates to the *hypothesis guessing* threat, where the subject might base their behaviour on their estimation of the hypothesis. Since the subjects knew the purpose of the experiment, they could have been able to precisely guess the hypothesis. However, any observable changes or behaviours of the subjects between the two models were observed, indicating that even if they would have correctly guessed the hypothesis, or made an assumption, it would not have affected the results.

The *evaluation apprehension* threat talks about how some people are afraid of being evaluated, and thus they will try to perform better just for the evaluation's sake. Even though the guidelines set in place would not have been able to fully eliminate this threat, they could have mitigated it by providing the subject with clear instructions on how to perform their task. Finally, there is the *experimenter expectancies* threat, which indicates that the experimenter can bias the results based on their expectations of the experiment. This can be mitigated by including people that have various expectations of the experiment. The subjects did all have different levels of experience working with AWS and could as well therefore also have had different expectations going into the experiment, which would have mitigated the threat.

Conclusion Validity. The threats regarding the conclusion validity revolve around whether the correct conclusions can be drawn from the experiment regarding the treatment and the outcome. The sample size was ten subjects, which could indicate a *low statistical power* threat where the true conclusions might not have been drawn. However, due to time constrictions, this threat could not have been mitigated further other than having included subjects with different levels of experience. The threat *reliability of measures* indicates that measures that depend on human judgement may not be the same if you perform the same measurements twice. This issue might depend on poor question wording, for example. In this experiment, the Efficiency metric depended on human input, meaning that if a subject were to perform the same test twice, the measurements would probably differ. The guidelines in place mitigated this threat somewhat by steering the subjects in a certain direction during each task. In contrast, the Complexity metric was an objective measure using lines of code which stayed the same throughout the experiment, meaning it would not change if the same subject did the test twice. The Risk metric was also an objective metric because it was the same throughout the experiment as well. As with most other relevant threats, the

guidelines used during the experiment would have mitigated the reliability threat regarding the Efficiency metric.

The threat called *reliability of treatment implementation* means that the implementation of the experimentation treatment is not similar between different subjects or occasions. This threat was mitigated through the use of the guidelines that helped the subject step through each task. Furthermore, each code that they were going to use for each task was completed beforehand, which meant that the subject only had to copy and paste the code into their respective destinations inside AWS.

Although the experiment procedure itself was controlled, the actual physical environment surrounding the subject was a lot less predictable. According to the *random irrelevancies in experimental setting* threat, elements outside the control of the experimental setting, such as noise or other interruptions, might impact the results negatively. A meeting room was booked for each subject that was able to perform the experiment on location which meant that the noise level was minimal and that there were no elements that would interrupt the session. However, for those subjects who could not attend in person, a distance meeting was booked and the subject could use a feature in the meeting program where they could control the actions of the host computer. Therefore, it was not possible for the experiment to control the physical environment in which the subject had chosen to perform the experiment during those distance meetings. The only aspect that could be managed was for the experiment leaders to sit in a meeting room themselves to minimise the noise level on their end. Additionally, there were irregularities during the experimentation sessions where there were small breaks or other interruptions which caused the experimentation to halt for some time. During those periods, the timer was stopped and resumed as soon as the session was ready again, which mitigated unnecessary timings.

Considering the subjects' backgrounds, they were all developers working at NIRA Dynamics. Therefore, due to their similar background, the group of subjects could be considered a homogeneous group. According to the *random heterogeneity of subjects* threat, this would suggest that individual differences among the subjects could have had less impact on the results than the experimentation treatment, which is a risk with heterogeneous groups. However, the group of people chosen to be test subjects was chosen only among NIRA employees and not a general population, which would reduce the external validity.

5 Results

This chapter will present the results, from the survey and experimentation, that this thesis has produced. It will also justify each iteration presented in the Method chapter.

5.1 Survey Results

This section will present the answers to the questions in the survey that was presented in Chapter 4.1. Not all the survey results are shown in this section, because the results that were either deemed to be confidential or not relevant enough were excluded. In total, 9 people answered the survey. All were employees at the company where this thesis performed its work at.

What is your main responsibility in your team? Out of the 9 employees who took part in the survey, 6 were developers and 3 had managerial positions inside the company, such as Scrum Master or Manager. In other words, 66.6% of the employees who answered were developers.

Do you think the problem was solved in the right way? In the survey, this question was asked in order to formulate a general opinion on how the current system worked at the company. Furthermore, this question was asked in regard to a previous question where the interviewee was asked how often they had a certain problem regarding the current access control, where they needed more access than their role(s) allowed. The results are illustrated in Figure 5.1 below, where the answers show that, among the interviewees, 44.4% thought that the problem was *not* solved in the right way, 22.2% thought it *was* solved in the right way, 11.1% were not sure, and 22.2% had not experienced said problem earlier. A follow-up question asked the interviewees to elaborate on why they chose the options *No* or *I'm not sure*. One of the answers stated that, while having too much access was a problem, there was no role that would yield the *right amount* of access. Another one stated that, if there were no security risks involved with a certain resource, it should be available. While a little bit less than half of the interviewees agreed that the access control was not handled in the right way, the general consensus of the elaborations, and a specific comment, is that it all depends on the situation and that there is no clear solution to solve everything.

The following Table 5.1 displays the answers to some of the most relevant questions that had a linear scale as an answer. In this scale, over several questions, the number 1 could mean: the interviewee did not agree, the time taken to perform a task was short, an action

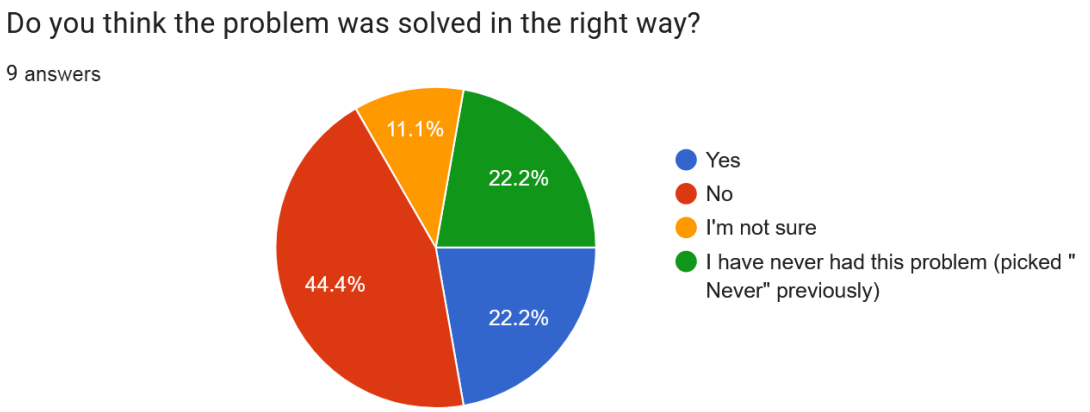

Figure 5.1: Pie chart illustrating the results of the question stated in the figure

Question	1	2	3	4	5
You think that the current system/workflow works well	0	33.3	55.6	11.1	0
Do you know which roles are available and what they do?	0	22.2	0	66.7	11.1
Do you feel like the names of the roles correlate to what they allow?	0	66.7	22.2	0	11.1
Do you feel that the Role object is currently used in a correct way?	11.1	22.2	33.3	33.3	0
How often do you need more access than what your role/roles allow? (Summarise comments (ask admin / architect for help))	55.6	44.4	0	0	0
How much time did this solution take?	25	0	75	0	0
How often is some further action needed for you to get access to a new resource?	77.8	11.1	11.1	0	0
How often do you/your team need to create new resources?	22.2	11.1	55.6	11.1	0

Table 5.1: Table consisting of the most relevant survey questions and their corresponding answers

did not occur often, or similar. In contrast, the number 5 meant: that the interviewee did agree, the time taken to perform a task was time-consuming, an action occurred often, or similar. The answers are displayed as a percentage number, representing how many of the interviewees picked each option.

Following the question **Do you feel that the Role object is currently used in a correct way?**, the survey prompted the interviewees to leave comments regarding the topic. Some comments included the idea of having one person per team that is able to delegate access to other team members, or that it should be clear which rights a user has if they have a certain role. Moreover, while one comment stated that the roles should be smaller and more specific, another stated that the roles should have broader permissions, rather than having more roles, which another comment agreed with, and added that it was better than fine-grained roles.

Following the question **How often do you need more access than what your role/roles allow?**, the survey once again prompted the interviewees to leave comments regarding the topic. There were fewer answers here, but in summary, they all gave the same answer: ask an administrator for help.

During the final segment of the survey, the interviewees were asked questions regarding the general workflow of AWS. One question in particular asked which steps they usually take when managing a policy in AWS. The answers ranged from looking at the documentation to

Policy 5.12 Updated Trust Policy for an IAM Role

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "Code1",
      "Effect": "Allow",
      "Principal": {
        "AWS": "*"
      },
      "Action": "sts:AssumeRole",
      "Condition": {
        "StringEquals": {
          "aws:ResourceTag/role": "${aws:PrincipalTag/<role>}"
        }
      }
    }
  ]
}
```

looking at previous policies or general solutions. Some even made minor changes in the existing infrastructure and executed the changes from there. When asked about how long these actions would usually take, the answers were evenly centred around the average time, where *short* and *long* time had 28.6% each, while 42.9% was in the middle.

5.2 Entity-centric Attribute- & Role-based Access Control Model

This section will summarise and justify the final decisions made from each iteration mentioned in Method 4.2, namely Iteration 1: Attributes, Iteration 2: Roles, and Iteration 3: Access Control.

The goal of Iteration 1 was to find a scalable and flexible solution using attributes (Tags), whilst also being able to have fine-grained access to objects on an individual level. While this was a sought-after property, as mentioned by Leitner et al. [17], it can sometimes be overly restrictive to have too fine-grained access as well. This made it increasingly important to have some sort of grouping permission for resources. However, since S3 did not support this for buckets or folders, the first and second solutions, namely the Public Read(List) and the two groupings, were not enough for an access control model. This left solutions three and four, namely the naming standard and the divided identity and resource policy solution. Whilst the naming standard did increase the complexity of the model as a whole, it only needed one policy to work. This meant that, if the access control would need to be changed in the future, it would only need to be changed in one place instead of two, which the fourth solution might have needed depending on the access control implemented. Therefore, the third solution, shown in Policy 4.5, was selected for Iteration 1, due to simpler manageability.

The goal of Iteration 2 was to add administrative purposes through roles, in addition to the scalability, flexibility, and fine-grained access control that attributes brought, to the model. Another important goal was also to allow cross-account access which was only possible through roles. Hence, the first policy created was the Trust Policy for each role, which handled who could assume it. However, whilst there was only one policy presented for this Trust Policy, see Policy 4.7, one flaw was found that had to be changed. Initially, for this policy to work, each user had to have the key *role* attached to them, however, as keys can only have one value, this constrained a user to only being able to access one role. Therefore, this policy received a minor change, see Policy 5.12, that switched the positions of *aws:PrincipalTag* and *aws:ResourceTag*, and changed *aws:PrincipalTag/role* into *aws:PrincipalTag/<role>*; <role> in

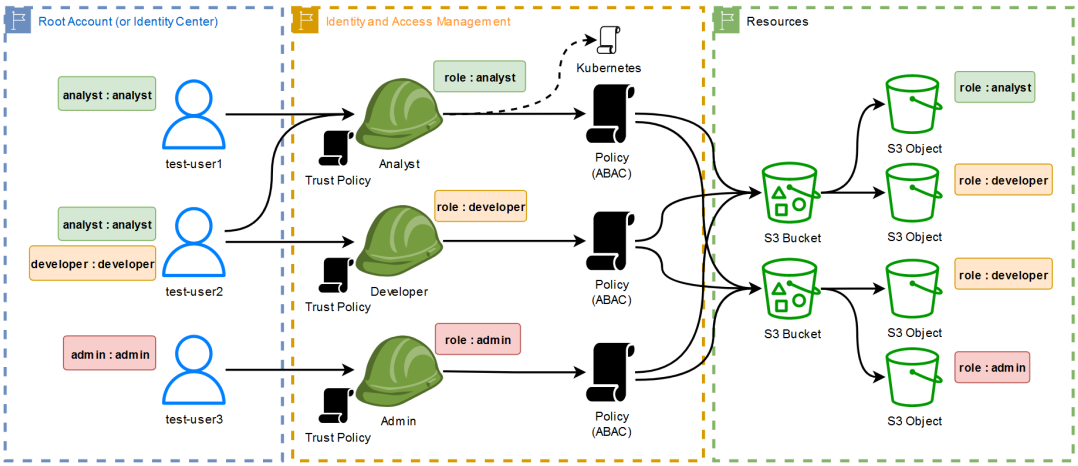

Figure 5.2: Illustration of the EARBAC model within AWS IAM

this case would be changed into the name of the role that is being set up. The last change meant that now, instead of matching a user's *role* key, a specific role key could be inserted. This made it possible for users to have multiple role keys, to be able to access multiple roles, instead of just one.

Furthermore, whilst the Trust Policy is a big part of Iteration 2, there was a policy needed for giving each role their corresponding access rights. For this matter, two solutions were created, namely one Identity-based policy for each resource and one Resource-based policy that specified what each role could do on that specific resource. Furthermore, the second solution also needed an Identity-based policy for the same reason as the fourth solution in Iteration 1. However, by sticking to the fact that using only one policy yielded simpler manageability in Iteration 1, the first solution was chosen, see Policy 4.8. Furthermore, in this iteration, the naming standard from Iteration 1 was dropped, because of its increased complexity, due to the roles now acting as groups for who could read or list buckets.

Finally, the goal of Iteration 3 was to, in a secure and efficient way, be able to manage the access control that had been set up from the previous iterations. To do so, two solutions were tested, namely outsourcing the access control responsibility to the roles in the role hierarchy and creating a new role whose sole purpose would be to manage the access control in the system. Whilst the second solution was similar to Uddin et al.'s [32] proposed solution, where they used a Process Owner role and multiple Security Coordinator roles, this solution became abstract and somewhat insecure. The reason for this was due to the uncertainty of knowing who should be able to assume a role that can change all the access rights within a system. Furthermore, by being able to change all access rights, the use of Tags was not needed, which resulted in less granularity when managing the system. The first solution, however, did not have these problems as the use of Tags allowed for more fine-grained access management whilst also adding another layer of management, namely changing Tags instead of policies. Therefore, the first solution was chosen for managing the proposed access control model moving forward.

In conclusion, all iterations introduced necessary functionalities into the proposed access control model, see Figure 5.2. Iteration 1 proposed the use of attributes (Tags), which allowed entities (users) to access resources. Iteration 2 proposed the addition of roles, which allowed groupings of the entities themselves. Furthermore, Iteration 3 proposed who and how the first two iterations were supposed to be managed moving forwards. However, whilst all of the iterations brought a new change or addition to the proposed access control model, they all had something in common, namely that the policies were always placed on the entities (users and roles) themselves. This meant that the policies were centralised to one location in AWS

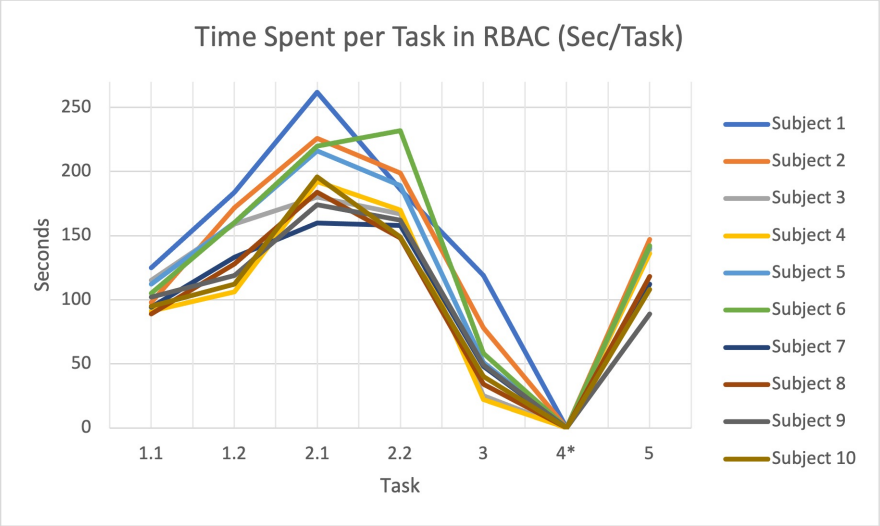

Figure 5.3: Chart illustrating the timings for the RBAC model

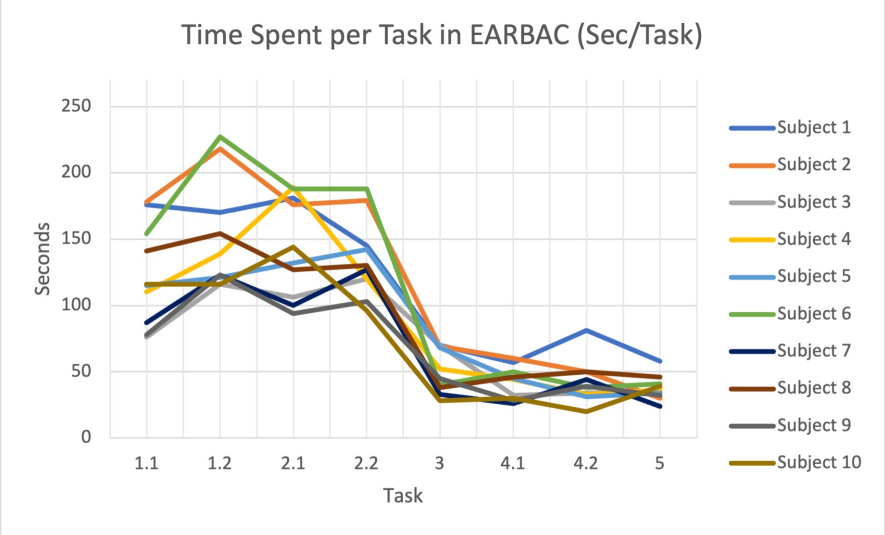

Figure 5.4: Chart illustrating the timings for the EARBAC model

IAM (in this case the roles) from where a user could manage the whole system, instead of multiple different places which might increase the risk of having overlapping access rights. It was for this specific reason that the proposed access control model got its name: The Entity-centric Attribute- & Role-based Access Control (EARBAC) model, illustrated previously in Figure 5.2. In this figure, it can be seen that users with certain Tags can access resources according to the implemented role hierarchy and all policies, for each resource, have been centralised around the roles for simpler management.

5.3 Experimentation Results

In this section, the results from the experimentation will be shown. Figure 5.3 shows the results of the experimentation on the RBAC model, while Figure 5.4 shows the results of the experimentation on EARBAC. In these following figures, the x-axis represents a task performed by a subject (NIRA employee) during the experimentation, while the y-axis represents how many seconds it took for that subject to perform the task. The legend to the right represents

Model	Lines of Code
RBAC	71
EARBAC	104

Table 5.2: Table consisting of the results from the complexity measurements

Model	Policy Changes After Setup
RBAC	4
EARBAC	0

Table 5.3: Table consisting of the results from the risk measurements

each subject and refers to the same subjects in both figures. Task 4 was not able to be performed using the RBAC model and this is represented by a timing of zero seconds in Figure 5.3. Tables showing the complete set of results are presented in Appendix C. Moreover, the results from the measurements for the Complexity metric are shown in Table 5.2, while Table 5.3 shows the results from the measurements for the Security metric. Each subject was able to provide feedback on the experimentation after they were done, and each subject's feedback is shown in Table 5.4 below.

Subject Nr.	Feedback	Preferred Model
1	The RBAC model quickly becomes more difficult to manage when systems grow	EARBAC
2	In this specific environment, EARBAC was better, however, it was difficult to get a good perspective of how it would work if the environment was larger	EARBAC
3	It was difficult to visualise if more side-by-side role hierarchies would become messy with EARBAC, however, the possibilities for it look great	EARBAC
4	Saw possibilities in managing EARBAC using scripts	EARBAC
5	EARBAC felt easier to understand	EARBAC
6	Whilst the RBAC model clearly shows who can do what (through ARNs), EARBAC gains another access management abstraction. This made EARBAC feel more fail-safe in comparison to the RBAC model	EARBAC
7	There might be an issue to integrate tagging (attributes) in some assisting systems outside of AWS (due to the absence of Tag support in some of these programs)	EARBAC
8	EARBAC, and tagging, felt clear and understandable	EARBAC
9	EARBAC feels very misrepresented since the RBAC model has been immensely simplified due to the copying of code directly instead of searching for each ARN before inserting it into the policy (which is how it has to be done). In a normal case, the timings for the RBAC model would be much slower in comparison to how little time EARBAC took	EARBAC
10	If one cannot use Tags on a specified granularity level (S3 only supports the lowest level, namely objects), then it is better to wait for support before changing the access control. May help to divide up the roles more, for example, resource:analyst, resource:developer, and resource:admin or similar	EARBAC

Table 5.4: Table consisting of the subjects' feedback from the experiment tasks

Subject Nr.	1	2	3	4	5	6	7	8	9	10
RBAC	169	153	131	119	145	153	118	117	116	117
EARBAC	126	129	80	97	92	125	77	98	73	80

Table 5.5: Table consisting of the average time it took for each subject to complete the tasks in RBAC and EARBAC

From the timing results in Appendix Tables C.2 and C.1, the variables used in the Paired t-test could be determined. The variables are displayed and calculated as follows:

$$t_0 = \frac{\overline{d}\sqrt{n}}{S_d}$$

$$S_d = \sqrt{\frac{\sum_{i=1}^{n}(d_i - \overline{d})^2}{n-1}}$$

where $d_i = x_i - y_i$. The x- and y-variables come from the paired samples: (x_1, y_1), (x_2, y_2), ..., (x_n, y_n), and n is the number of samples, which in this case is 10 (number of test subjects). The paired samples represent the average time spent per subject on each task for each model. For example, the variable x_1 is calculated from a subject's total time spent on all tasks for RBAC divided by the number of tasks, while y_1 represents the same but while using EARBAC instead, these values can be seen in Table 5.5. By inserting the paired samples into d, it resulted in: $d = \{-13, -32, -6, -34, -4, -29, -16, -38, -14, -20\}$, with the median $\overline{d} = -18$. Finally, this resulted in the t-value $t_0 = -4.627$

6 Experimentation Evaluation

In this chapter, the results will be evaluated to see how EARBAC performed against an RBAC model. The results presented in Figure 5.3, and Figure 5.4 show that the EARBAC model was more efficient, with an average time of *98* seconds to complete each task per subject was, whereas RBAC had an average of *134* seconds. As expected, the efficiency results of the EARBAC model match the expected results of a proposed access control model utilising the benefits of both ABAC and RBAC. It should be noted that Task 4.1 was excluded from the calculations, with the reason being that that specific task illustrated a case where the results ended up including an error, which was only supposed to educate the subject before solving it in *Task 4.2*. Therefore, by including the timings from Task 4.1, the numbers would not have represented the optimal solution in that case, which was the point of all the other tasks. Furthermore, the RBAC model could not perform Task 4.1, whereas EARBAC could, and in an efficient manner as well. In other words, a developer was able to efficiently perform an administrative task, within their access rights, without having to consult an actual admin. In a realistic scenario, this would have saved a lot of time. Also, the last column of the feedback results, shown in Table 5.4, shows that the preferred access control model among the subjects was EARBAC.

It was expected of the complexity metric to be higher for EARBAC than for the RBAC model. The reason for this was because EARBAC relied heavily on the ABAC model which in turn yielded a higher computing overhead, as explained in Chapter 2.5.5. Moreover, by determining the complexity only by counting the lines of code in finished policies, a trend could not be observed from several measurements due to the metric being measured only once. However, since the policy changes in EARBAC were only needed during setup, whereas they were needed all throughout the RBAC model's run time, this thesis estimate that the complexity metric will not increase much, if at all, during the run time of EARBAC; compared to the RBAC model, where the complexity will likely keep on increasing during run time, EARBAC shows promising results. Furthermore, with the access control being less complex during run time by using EARBAC, the developers will more likely understand how to perform certain actions without having to consult an admin.

Finally, the results of the security metric, presented in Table 5.3, show that EARBAC has higher security than the RBAC model. This is due to the fact that no policy changes had to be made after setup to increase selected users' access rights during specific tasks. Furthermore, because there was no need to edit or add a policy using EARBAC, a user with the developer

role was able to perform this task. This is one of the observed benefits of the model, that developers can make administrative changes without affecting the overall policy structure.

In RBAC, if a developer without precise knowledge of the system has to make changes to the system, given that an admin is not able to help them and they gain temporary admin privileges, it can lead to negative effects in which the wrong users gain access or that authorised users get locked out of necessary resources. It is, however, still possible for an admin to make mistakes while editing the policies, in both models. Therefore, by relying on policies that dictate that roles with certain Tags gain access to resources with the same Tags, it minimises the risk of mistakes by having the users Tag resources instead of making policy changes each time they need to update access rights. This was also supported by the subjects' feedback, presented in Table 5.4, where EARBAC felt more fail-safe than the RBAC model.

To reject the null hypotheses, the absolute value of the t-value, $|t_0|$, has to be larger than $t_{\alpha/2,n-1} = 2.262$. This experimentation reaches the same value of $t_{\alpha/2,n-1}$ as Wohlin et al. [33] do in a mock experimentation because of a similar setup and number of test subjects used. By proceeding with the results from Chapter 5.3, the following relation can be constructed: $|t_0| = 9.836 > 2.262 = t_{\alpha/2,n-1}$. Since this relation is true, it means that the null hypotheses can be rejected. However, this does *not* mean that one access control model is better than the other, just by rejecting the null hypotheses, but what it can mean is that enough samples were gathered and varied enough so that the results are reliable.

7 Discussion

This chapter will provide insight and criticism into the work that has been performed in order to complete this thesis. This chapter will mainly go through the Method chapter, as well as discuss the work in a wider context.

7.1 Method Discussion

The survey yielded insight into what slowed down the company's current access control system and what could be improved, but after a few sessions of conducting the survey, no additional insight was gained and each new session started to feel a bit repetitive. In hindsight, fewer survey sessions could have been held, which in turn would have yielded more time that could have been spent on refining other parts of the thesis, such as the experimentation. Furthermore, if survey questions that proved not to be useful were to be ignored, more time could have been saved. For example, questions such as the ones related to how an employee performs certain tasks did not have to be in the survey at all as these especially became repetitive. Additionally, the survey was held during the time frame of approximately two weeks. However, nearing the end of those two weeks, the company hosted a workshop on AWS to gain further insight into different areas of AWS. This could have influenced the remaining survey participants' answers since they received more knowledge about the possibilities within AWS, compared to the other survey participants who only had their own prior experiences when answering the questions. Ultimately, the two different sets of participants were not distinguishable from one another, however, the possibility still remains. Finally, if one would wish to conduct the survey themselves, all the questions used have been presented in Appendix A.

Since the development of a new access control model takes both existing knowledge as well as trial and error, it might be difficult to understand each step taken during the iterations when developing the proposed access control model. However, since every step has been implemented and explained, in as much detail as possible, it would not be difficult to recreate the steps taken if one were to use the AWS platform. It might not, however, be as simple to implement these iterations, down to the exact policy code, in other systems. A reason for this is that the AWS platform might use some features that other systems do not use, such as attributes and roles, even though these two are common in today's access manage-

ment systems. Therefore, if the AWS platform were to be used to recreate this solution, the replicability would be high since all resources used have been supplied for each step.

Furthermore, as explained during the experimentation method, the initial experiment was somewhat biased towards the RBAC model. The solution to this was to retroactively update the measurements of the RBAC model. However, since these measurements were only done by two new subjects, as well as only on specific steps, it could have resulted in a worse representation of the actual model since more subjects were used for the initial experimentation. To reduce the worst-case scenario for these two subjects, they both performed the test twice. The reason for this was that the first measurement, for each subject, was expected to be a bit higher (slower performance) whilst the second measurement became lower (faster performance) since they had already performed the steps once already. This made the average between the two measurements more unbiased, in comparison to if two different subjects were to perform the same test once. In conclusion, this would have reduced the effect of a possible outlier when calculating the average time for each task. On the other hand, by instead using time that could have been saved from a shorter survey, more time could have been spent on creating a more accurate representation of the RBAC model from the beginning. That way, the timings would most likely have resulted in even more accurate numbers for the RBAC model, since the test subjects would have performed the additional RBAC-specific steps themselves, which would have made the efficiency metric more reliable.

It is, however, more difficult to argue that the complexity and security metrics accurately represent the intended measurements, since they were only measured once and are not as clear on what insight they provide. However, seeing as this thesis has conducted more of a qualitative study rather than a quantitative one, it would be difficult either way to find a metric for those two aspects that accurately represent the intended purpose. To summarise, the metrics used in this thesis have pretty good validity but could be further refined to be more accurate.

Furthermore, since the experimentation was conducted in an already set up environment, with finished code and tasks to be carried out by the subjects, it would be simple to recreate this experimentation for any other scenario. This would indicate good replicability of the experimentation if the same resources were to be used and the subjects were selected in a similar fashion. Moreover, because the experimentation was conducted in a company where each subject had a different level of experience, and because of the structured experimentation steps, the efficiency metric would most likely have been similar for other experiments following the same procedure and setup. Additionally, since the complexity and security metrics depended on prewritten code and the AWS system only, not human input, they would have been the same as well. This would indicate good reliability because the results would most likely be similar in any other scenario following the same steps as in this thesis.

A majority of the sources have originated from reputed publishers which include IEEE, Springer, and ACM, and are therefore considered trusted sources. However, even though there were varied sources, most of them were published by IEEE and found on IEEE-Xplore. An effort could have been made to use other publishers' search engines as well to find more varied results or to use the Google Scholar search engine more. However, while many of the trusted sources that were found yielded relevant and supporting claims to how access control works, it has to be taken into account that Ravi Sandhu has been an author or co-author to seven of these papers. This might result in some biased claims regarding access control, however, since Sandhu's previous work on access control[1] is extensive, this was seen as a benefit instead of an issue.

Moreover, a portion of the sources are from AWS's own web page, including documentation and product descriptions. Therefore, the information presented on those web pages could be seen as biased, and more studies on AWS, not by AWS themselves, could have provided useful insight into how the system works. Additionally, an AWS architect (working for

[1] https://profsandhu.com/

AWS) was consulted through NIRA to help with certain problems. While this help could also have been seen as biased, due to the fact that AWS might want their customers to use more of their services, the questions pertaining to the problem were specific and technical. This meant that there was no incentive to sell more services since the questions were specified within IAM, which was already in use before this thesis started its work.

7.2 The Work in a Wider Context

The result of this thesis is meant to improve security, among others. Therefore, it may positively impact a user or company that decides to use the EARBAC model in their system. However, since the model has only been tested in a controlled environment, there might be unknown weaknesses that might occur if proper precautions are not taken upon implementation and management. Furthermore, since cloud computing usually indicates decentralised storage, it can be seen as a privacy risk when storing data on those servers. The reason for this is that those servers are not owned by the user itself but by a company that might have obligations towards authorities when it comes to handing over data. An access control model, however secure, would not have protected against this scenario.

8 Conclusion

In today's industry, there are many access control models in place that help with authorisation in cloud computing. This thesis' proposed access control model, EARBAC, helps companies and developers using AWS to apply a fine-grained access control that is both flexible and scalable for their resources. EARBAC uses Tags in addition to roles to gain another access control level that increases security and efficiency, in comparison to RBAC which only relies on administrative management.

Research Question 1: What counts as an efficient security policy strategy?

As seen in the Related Work chapter, many authors write about access control models focusing on having a flexible and manageable model in addition to the main aspect which is security. Furthermore, since all of these aspects are subject to how much a system grows and how complex it becomes, scalability also has to be taken into account. Moreover, when systems become larger and more complex, they also become more difficult to manage which rolls back to the need for flexibility and manageability. Therefore, in creating an access control model that facilitates these aspects, the process of managing this model should be simple and easy to understand for the average user. No complex rules or syntax should be necessary to learn beforehand, and it should not take up much of the developer's time to update access rights for a lesser role in the hierarchy.

Research Question 2: How can a balance be found between security and developer efficiency utilising the strategy found in RQ_1 for AWS IAM?

By removing the responsibility of making policy changes from the average developer, the risk of making mistakes decreases. Furthermore, by utilising a strategy such as tagging, a developer can increase a lower role's access rights without having to consult an administrative role, eliminating the risk of the developer taking unnecessary shortcuts in their workflow, for example by assuming an overqualified- or administrative role. Furthermore, if an admin becomes overloaded with work and access requests in the system, they might make uninformed or quick solutions that endanger the system through the use of their increased privileges. Therefore, by taking the load off of the admin and placing it on the developer, the admin will not have to step in as much and help other roles with their tasks and can spend more time on making informed and correct solutions.

Research Question 3: How can AWS IAM be used to create an access control model that uses the advantages of both Role-based Access Control and Attribute-based Access Control?

As seen in the proposed access control model EARBAC, policies can be used to connect and manage AWS resources of any kind. When it comes to ABAC, AWS IAM offers resource metadata, that can be used as attributes, and are called Tags. Tags consist of key-value pairs that, together with policies, help users build and manage access control models for their resources in AWS. When it comes to RBAC, AWS IAM offers roles which group users and allow for cross-account access. These roles also have resource metadata which makes them applicable for tagging. In combination, these two features can be used to access any role within any AWS account by only applying the correct Tag on users, making it even more simple to access resources divided into different AWS accounts.

A Survey Questions

No.	Question	Answer type	M/O
1	Which team(s) are you assigned to?	Checkboxes	M
2	What is your main responsibility in **your** team?	Short answer	M
3	What is your main responsibility outside of the team?	Short answer	O
4	You think that the current system/workflow works well	Linear scale (1-5)	M
5	In general, is there anything else at Nira that has complicated your work regarding AWS? (That has not been brought up in the survey)*	Paragraph	O

* This question appears last in the survey but belongs to this group

Table A.1: General questions

No.	Question	Answer type	M/O
6	Do you know which roles are available and what they do?	Linear scale (1-5)	M
7	Do you feel like the names of the roles correlate to what they allow?	Linear scale (1-5)	M
8	Do you feel like the Role object is currently used in a correct way?	Linear scale (1-5)	M
9	How often do you need more access than what your role/roles allow(s)?	Linear scale (1-5)	M
10	How did you solve this?	Paragraph	O
11	How much time did this solution take?	Linear scale (1-5)	O
12	Do you think the problem was solved in the right way?	Multi answer	M
13	If you answered *No* or *I'm not sure*, please elaborate on why	Paragraph	O

Table A.2: Questions pertaining to Roles in AWS

No.	Question	Answer type	M/O
14	How often do you need access to another team's resources?	Linear scale (1-5)	M
15	How often is some further action needed for you to get access to a new resource?	Linear scale (1-5)	M
16	How much time does this action usually take?	Linear scale (1-5)	O
17	How often do you/your team need to create new resources?	Linear scale (1-5)	M
18	How much time do you usually need to create new resources?	Linear scale (1-5)	O

Table A.3: Questions pertaining to Resources / Services in AWS

No.	Question	Answer type	M/O
19	How often do you have to add, delete or modify a ...*	Multiple choice grid*	M
20	When managing a **User**, which steps do you take?	Paragraph	O
21	How much time does this action usually take?	Linear scale (1-5)	O
22	When managing a **Role**, which steps do you take?	Paragraph	O
23	How much time does this action usually take?	Linear scale (1-5)	O
24	When managing a **Policy**, which steps do you take?	Paragraph	O
25	How much time does this action usually take?	Linear scale (1-5)	O
26	When managing a **Resource**, which steps do you take?	Paragraph	O
27	How much time does this action usually take?	Linear scale (1-5)	O
28	When creating a **New Project**, which steps do you take?	Paragraph	O
29	How much time does this action usually take?	Linear scale (1-5)	O
30	When adding **Customer Access**, which steps do you take?	Paragraph	O
31	How much time does this action usually take?	Linear scale (1-5)	O

*The answers to this question are represented as several rows: *User*, *Role*, *Policy*, *Resource*, *New Project*, and *Customer Access*

Table A.4: Questions pertaining to the general Workflow in AWS

B Experimentation Guidelines

As stated previously, the tasks were timed. However, the steps marked with **Test:** were not timed because those steps are essentially present in order to just evaluate whether the previous steps were performed correctly. Thus, they did not need to be timed.

B.1 Guidelines for experimentation with Role-based Access Control

Task 1.1: Setup Assume Role for User (starting as admin)

1. Navigate to *IAM -> Roles -> analyst -> Trust relationships -> Edit trust policy*
2. Insert Policy 1 E.14 (with updated user-arn) and press *Update policy*
3. **Test:** Change User from *Admin* to *Test User*
4. **Test:** Assume Analyst Role

Task 1.2: Setup Assume Role for User (starting as admin)

1. Repeat the steps from the previous task for *Developer* as well
2. Now also add three new users *test-user2* and *test-user3* to the Developer Role, using Policy 2 E.15
3. **Test:** Change User from *Admin* to *Test User*
4. **Test:** Assume Developer Role

Task 2.1: Create Identity-based Policy for Analyst Role (starting as admin)

1. Navigate to *IAM -> Roles -> analyst -> Add permissions -> Attach policies -> Create policy -> JSON*
2. Insert Policy 3 E.16 with updated stack name and press *Next*
3. Below *Policy Name*, write: *rbac-analyst*, and then press *Create policy*
4. Navigate to *Roles -> analyst -> Add permissions -> Attach policies*

5. Check the box for rbac-analyst, scroll down and press *Add permissions*
6. **Test:** Change User from *Admin* to *Test User*
7. **Test:** Assume Analyst Role
8. **Test:** Navigate to *Amazon S3* -> *Buckets* -> *test-stack*
9. **Test:** Check the box for analyst.txt and press *Download*
10. **Test:** Check the box for developer.txt and press *Download*

Task 2.2: Create Identity-based Policy for other roles (starting as admin)

1. Repeat the steps from the previous task for *Developer* using Policy 4 E.17
2. **Test:** Change User from *Admin* to *Test User*
3. **Test:** Assume Developer Role
4. **Test:** Navigate to *Amazon S3* -> *Buckets* -> *test-stack*
5. **Test:** Check the box for analyst.txt and press *Download*
6. **Test:** Check the box for developer.txt and press *Download*

Task 3: Developer writes Object new_object.txt

1. **Test:** Assume Developer Role
2. **Test:** Navigate to *Amazon S3* -> *Buckets* -> *test-stack*
3. **Test:** Try to write new_object.txt, by pressing *Upload* -> *Add files* -> *new_object.txt* -> *Open* -> *Upload*
4. **Test:** Change User from *Test User* to *Admin*
5. Now try to give the Developer Role write access by using Policy 5, E.18
6. **Test:** Assume Developer Role
7. **Test:** Navigate to *Amazon S3* -> *Buckets* -> *test-stack*
8. **Test:** Try to write new_object.txt, by pressing *Upload* -> *Add files* -> *new_object.txt* -> *Open* -> *Upload*

Task 4: Developer updates Object new_object.txt

1. **Test:** Assume Analyst Role
2. **Test:** Navigate to *Amazon S3* -> *Buckets* -> *test-stack*
3. **Test:** Check the box for new_object.txt and press *Download*
4. **Test:** Assume Developer Role
5. **Test:** Navigate to *IAM* -> *Roles*

Task 5: Admin updates Object for Analyst to read

1. Assume Admin Role
2. Now try to give the Analyst Role access to *Download* new_object.txt using Policy 6, E.19
3. **Test:** Assume Analyst Role
4. **Test:** Navigate to *Amazon S3* -> *Buckets* -> *test-stack*
5. **Test:** Check the box for new_object.txt and press *Download*

Reset Environment

B.2 Guidelines for experimentation with the Proposed Model

Task 1.1: Setup Assume Role for User (starting as admin)

1. Navigate to *IAM -> Roles -> analyst -> Trust relationships -> Edit trust policy*
2. Insert Policy 1 E.20 (with updated Tag) and press *Update policy*
3. Navigate to *Tags -> Manage tags -> Add tag*
4. Insert *role* as key and *analyst* as value, and press *Save changes*
5. Navigate to *Users -> test-user -> Tags -> Manage tags -> Add new tag*
6. Insert *analyst* as both key and value, and press *Save changes*
7. **Test:** Change User from *Admin* to *Test User*
8. **Test:** Assume Analyst role

Task 1.2: Setup Assume Role for User (starting as admin)

1. Repeat the steps from the previous task for *Developer*, using Policy 1 E.20 again
2. Now also add three new users *test-user2* and *test-user3* to the Developer Role
3. **Test:** Change User from *Admin* to *Test User*
4. **Test:** Assume Developer Role
5. **Test:** Assume Admin Role

Task 2.1: Create Identity-based Policy for the Analyst Role (starting as admin)

1. Navigate to *IAM -> Roles -> analyst -> Add permissions -> Attach policies -> Create policy -> JSON*
2. Insert Policy 2 E.21 and press *Next*
3. Below *Policy name*, write: *earbac-analyst*, and then press *Create policy*
4. Navigate to *Roles -> analyst -> Add permissions -> Attach policies*
5. Check the box for earbac-analyst, scroll down and press *Add permissions*
6. Navigate to *Amazon S3 -> Buckets -> test-stack -> analyst.txt*
7. Scroll down to *Tags* and press *Edit -> Add tag*
8. Insert *role* as key and *analyst* as value, and press *Save changes*
9. **Test:** Change User from *Admin* to *Test User*
10. **Test:** Assume Analyst Role
11. **Test:** Navigate to *Amazon S3 -> Buckets -> test-stack*
12. **Test:** Check the box for analyst.txt and press *Download*
13. **Test:** Check the box for developer.txt and press *Download*

Task 2.2: Create Identity-based Policy for other roles (starting as admin)

1. Repeat the steps from the previous task for *Developer*, using Policy 3 E.22

2. **Test:** Change User from *Admin* to *Test User*
3. **Test:** Assume Developer Role
4. **Test:** Navigate to *Amazon S3* -> *Buckets* -> *test-stack*
5. **Test:** Check the box for analyst.txt and press *Download*
6. **Test:** Check the box for developer.txt and press *Download*

Task 3: Developer writes Object new_object.txt

1. **Test:** Assume Developer Role
2. **Test:** Navigate to *Amazon S3* -> *Buckets* -> *test-stack*
3. **Test:** Try to write new_object.txt, by pressing *Upload* -> *Add files* -> *new_object.txt* -> *Open*
4. **Test:** Navigate to *Properties*, scroll down and press *Add tag*
5. **Test:** Insert *role* as key and *developer* as value, and press *Upload*
6. **Test:** Change User from *Test User* to *Admin*
7. Now try to give the Developer Role write access by using Policy 4 E.23
8. **Test:** Change User from *Admin* to *Test User*
9. **Test:** Navigate to *Amazon S3* -> *Buckets* -> *test-stack*
10. **Test:** Try to write new_object.txt, by pressing *Upload* -> *Add files* -> *new_object.txt* -> *Open*
11. **Test:** Navigate to *Properties*, scroll down and press *Add tag*
12. **Test:** Insert *role* as key and *developer* as value, and press *Upload*

Task 4.1: Developer updates Object new_object.txt

1. **Test:** Assume Analyst Role
2. **Test:** Navigate to *Amazon S3* -> *Buckets* -> *test-stack*
3. **Test:** Check the box for new_object.txt and press *Download*
4. Assume Developer Role
5. Navigate to *Amazon S3* -> *Buckets* -> *test-stack* -> *new_object.txt*
6. Scroll down to *Tags* and press *Edit*
7. Change value of *role* from *developer* to *analyst*, and press *Save changes*
8. **Test:** Assume Analyst Role
9. **Test:** Navigate to *Amazon S3* -> *Buckets* -> *test-stack*
10. **Test:** Check the box for new_object.txt and press *Download*

Task 4.2: Admin restricts Developer tagging permissions

1. **Test:** Assume Developer Role
2. **Test:** Navigate to *Amazon S3* -> *Buckets* -> *test-stack* -> *admin2.txt*

3. **Test:** Scroll down to *Tags* and press *Edit*
4. **Test:** Change value of *role* from *admin* to *developer*, and press *Save changes*
5. **Test:** Change User from *Test User* to *Admin*
6. Now try to restrict the Developer Role's update access by using Policy 5 E.24
7. **Test:** Assume Developer Role
8. **Test:** Navigate to *Amazon S3* -> *Buckets* -> *test-stack* -> *admin.txt*
9. **Test:** Scroll down to *Tags* and press *Edit*
10. **Test:** Change value of *role* from *admin* to *developer*, and press *Save changes*

Task 5: Admin updates Object for Analyst to read

1. **Test:** Assume Analyst Role
2. **Test:** Navigate to *Amazon S3* -> *Buckets* -> *test-stack*
3. **Test:** Check the box for developer.txt and press *Download*
4. Assume Admin Role
5. Now try to give the Analyst Role access to *Download* developer.txt
6. **Test:** Assume Analyst Role
7. **Test:** Navigate to *Amazon S3* -> *Buckets* -> *test-stack*
8. **Test:** Check the box for developer.txt and press *Download*

C Experimentation Results (complete tables)

Subject Nr.	Task 1.1	Task 1.2	Task 2.1	Task 2.2	Task 3	Task 4	Task 5
1	125s	184s	262s	186s	119s	N/A	140s
2	98s	172s	226s	199s	78s	N/A	147s
3	115s	159s	180s	167s	25s	N/A	139s
4	91s	106s	192s	170s	22s	N/A	136s
5	112s	160s	216s	189s	51s	N/A	141s
6	105s	160s	220s	232s	58s	N/A	142s
7	94s	133s	160s	158s	48s	N/A	112s
8	89s	128s	184s	148s	34s	N/A	118s
9	102s	119s	174s	162s	49s	N/A	89s
10	95s	112s	196s	149s	40s	N/A	108s

Table C.1: Table consisting of the results from the experiment for the RBAC model

Subject Nr.	Task 1.1	Task 1.2	Task 2.1	Task 2.2	Task 3	Task 4.1*	Task 4.2	Task 5
1	176s	170s	181s	145s	70s	57s	81s	58s
2	178s	218s	176s	179s	69s	60s	50s	30s
3	76s	116s	106s	120s	70s	32s	34s	35s
4	110s	139s	189s	120s	52s	44s	33s	36s
5	115s	121s	132s	142s	68s	45s	31s	34s
6	154s	227s	188s	188s	40s	50s	38s	41s
7	87s	123s	100s	127s	33s	26s	44s	24s
8	141s	154s	127s	130s	38s	46s	50s	46s
9	78s	123s	94s	103s	45s	28s	39s	32s
10	116s	116s	144s	96s	28s	30s	20s	39s

Table C.2: Table consisting of the results from the experiment for EARBAC

D Parameter/NotParameter Table

Parameter	Effect (Deny)	Effect (Allow)
Principal	Explicitly Deny specified Principal(s)	Allow specified Principal(s)
NotPrincipal	(Not Advised)[1]	(Not Advised)[1]
Action	Explicitly Deny specified Action(s)	Allow specified Action(s)
NotAction	Explicitly Deny all Actions except the specified Action(s), however, specified Action(s) still have to be allowed from a separate Statement or Policy	Allow all Actions except the specified Action(s) (not Explicit Deny)
Resource	Explicitly Deny some Action(s) on specified Resource(s)	Allow some Action(s) on specified Resource(s)
NotResource	Explicitly Deny some Action(s) to all Resources except for the specified Resource(s)	**(Not Advised / Unsecure)**[2] Allow some Action(s) to all Resources except for the specified Resource(s)

Table D.1: Table of parameters and their combination of effects

[1]https://docs.aws.amazon.com/IAM/latest/UserGuide/reference_policies_elements_notprincipal.html

[2]https://docs.aws.amazon.com/IAM/latest/UserGuide/reference_policies_elements_notresource.html

E JSON Code

E.1 General

Policy E.13 Identity-based Policy for a Developer Role (S3, EC2, EKS, Other Resources)

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowsReadWriteUpdateDeleteForS3Resources",
      "Effect": "Allow",
      "Action": [
        "s3:Read",
        "s3:Write",
        "s3:Update",
        "s3:Delete"
      ],
      "Resource": "*",
      "Condition": {...}
    },
    {
      "Sid": "AllowsReadWriteUpdateDeleteForEC2Resources",
      "Effect": "Allow",
      "Action": [
        "ec2:Read",
        "ec2:Write",
        "ec2:Update",
        "ec2:Delete"
      ],
      "Resource": "*",
      "Condition": {...}
    },
    {
      "Sid": "AllowsReadWriteUpdateDeleteForEKSResources",
      "Effect": "Allow",
      "Action": [
        "eks:Read",
        "eks:Write",
        "eks:Update",
        "eks:Delete"
      ],
      "Resource": "*",
      "Condition": {...}
    },
    {
      "Sid": "AllowsReadWriteUpdateDeleteForOtherResources",
      ...
    }
  ]
}
```

E.2 Experimentation - Role-based Access Control

Policy E.14 Experimentation Policy 1 for Role-based Access Control

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "Policy1",
      "Effect": "Allow",
      "Principal": {
        "AWS": [
          "arn:aws:iam::<account-1>:user/<test-user>"
        ]
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

Policy E.15 Experimentation Policy 2 for Role-based Access Control

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "Policy2",
      "Effect": "Allow",
      "Principal": {
        "AWS": [
          "arn:aws:iam::<account-1>:user/<test-user>",
          "arn:aws:iam::<account-1>:user/<test-user2>",
          "arn:aws:iam::<account-1>:user/<test-user3>",
          "arn:aws:iam::<account-2>:user/<other-user>"
        ]
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

Policy E.16 Experimentation Policy 3 for Role-based Access Control

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "Policy3",
      "Effect": "Allow",
      "Action": [
        "s3:ListAllMyBuckets",
        "s3:GetBucketLocation",
        "s3:ListBucket"
      ],
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": "s3:GetObject",
      "Resource": "arn:aws:s3:::<test-stack>/analyst.txt"
    }
  ]
}
```

Policy E.17 Experimentation Policy 4 for Role-based Access Control

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "Policy4",
      "Effect": "Allow",
      "Action": [
        "s3:ListAllMyBuckets",
        "s3:GetBucketLocation",
        "s3:ListBucket"
      ],
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": "s3:GetObject",
      "Resource": [
        "arn:aws:s3:::<test-stack>/developer.txt",
        "arn:aws:s3:::<test-stack>/analyst.txt"
      ]
    }
  ]
}
```

Policy E.18 Experimentation Policy 5 for Role-based Access Control

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "Policy5",
      "Effect": "Allow",
      "Action": [
        "s3:ListAllMyBuckets",
        "s3:GetBucketLocation",
        "s3:ListBucket"
      ],
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": "s3:GetObject",
      "Resource": [
        "arn:aws:s3:::<test-stack>/developer.txt",
        "arn:aws:s3:::<test-stack>/analyst.txt"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "s3:PutObject"
      ],
      "Resource": "*"
    }
  ]
}
```

Policy E.19 Experimentation Policy 6 for Role-based Access Control

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "Policy6",
      "Effect": "Allow",
      "Action": [
        "s3:ListAllMyBuckets",
        "s3:GetBucketLocation",
        "s3:ListBucket"
      ],
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": "s3:GetObject",
      "Resource": [
        "arn:aws:s3:::<test-stack>/analyst.txt",
        "arn:aws:s3:::<test-stack>/new_object.txt"
      ]
    }
  ]
}
```

E.3 Experimentation - Proposed Model

Policy E.20 Experimentation Policy 1 for Proposed Model

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "Policy1",
      "Effect": "Allow",
      "Principal": {
        "AWS": "*"
      },
      "Action": "sts:AssumeRole",
      "Condition": {
        "StringEquals": {
          "aws:ResourceTag/role": "${aws:PrincipalTag/<role>}"
        }
      }
    }
  ]
}
```

Policy E.21 Experimentation Policy 2 for Proposed Model

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "Policy2",
      "Effect": "Allow",
      "Action": [
        "s3:ListAllMyBuckets",
        "s3:GetBucketLocation",
        "s3:ListBucket"
      ],
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": "s3:GetObject",
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "s3:ExistingObjectTag/role": [
            "<role>"
          ]
        }
      }
    }
  ]
}
```

Policy E.22 Experimentation Policy 3 for Proposed Model

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "Policy3",
      "Effect": "Allow",
      "Action": [
        "s3:ListAllMyBuckets",
        "s3:GetBucketLocation",
        "s3:ListBucket",
        "s3:GetObjectTagging"
      ],
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "s3:GetObject"
      ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "s3:ExistingObjectTag/role": [
            "developer",
            "analyst"
          ]
        }
      }
    }
  ]
}
```

Policy E.23 Experimentation Policy 4 for Proposed Model

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "Policy4",
      "Effect": "Allow",
      "Action": [
        "s3:ListAllMyBuckets",
        "s3:GetBucketLocation",
        "s3:ListBucket",
        "s3:GetObjectTagging"
      ],
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "s3:GetObject"
      ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "s3:ExistingObjectTag/role": [
            "developer",
            "analyst"
          ]
        }
      }
    },
    {
      "Effect": "Allow",
      "Action": [
        "s3:PutObject",
        "s3:PutObjectTagging"
      ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "s3:RequestObjectTag/role": [
            "analyst",
            "developer",
            "admin"
          ]
        }
      }
    }
  ]
}
```

Policy E.24 Experimentation Policy 5 for Proposed Model

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "Policy5",
      "Effect": "Allow",
      "Action": [
        "s3:ListAllMyBuckets",
        "s3:GetBucketLocation",
        "s3:ListBucket",
        "s3:GetObjectTagging"
      ],
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "s3:GetObject"
      ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "s3:ExistingObjectTag/role": [
            "developer",
            "analyst"
          ]
        }
      }
    },
    {
      "Effect": "Allow",
      "Action": [
        "s3:PutObject"
      ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "s3:RequestObjectTag/role": [
            "analyst",
            "developer",
            "admin"
          ]
        }
      }
    },
    {
      "Effect": "Allow",
      "Action": [
        "s3:PutObjectTagging"
      ],
      "Resource": "*",
      "Condition": {
        "StringNotLike": {
          "s3:ExistingObjectTag/role": "admin"
        }
      }
    }
  ]
}
```

Bibliography

[1] Ijaz Ahmad, Madhusanka Liyanage, Suneth Namal, Mika Ylianttila, Andrei Gurtov, Marcus Eckert, Thomas Bauschert, Zoltan Faigl, Laszlo Bokor, Ece Saygun, Hasan Anıl Akyıldız, Oscar Lopez Perez, Mikel Uriarte Itzazelaia, Berna Ozbek, and Aydın Ulas. "New concepts for traffic, resource and mobility management in software-defined mobile networks". In: *2016 12th Annual Conference on Wireless On-demand Network Systems and Services (WONS)*. 2016, pp. 1–8.

[2] Gail-Joon Ahn and Ravi Sandhu. "Role-Based Authorization Constraints Specification". In: *ACM Trans. Inf. Syst. Secur.* 3.4 (Nov. 2000), pp. 207–226. ISSN: 1094-9224. DOI: 10.1145/382912.382913. URL: https://doi.org/10.1145/382912.382913.

[3] Khalid Albulayhi, Abdullah Abuhussein, Faisal Alsubaei, and Frederick T. Sheldon. "Fine-Grained Access Control in the Era of Cloud Computing: An Analytical Review". In: *2020 10th Annual Computing and Communication Workshop and Conference (CCWC)*. 2020, pp. 0748–0755. DOI: 10.1109/CCWC47524.2020.9031179.

[4] Safwa Ameer, James Benson, and Ravi sandhu. "Hybrid Approaches (ABAC and RBAC) Toward Secure Access Control in Smart Home IoT". In: *IEEE Transactions on Dependable and Secure Computing* (2022), pp. 1–18. DOI: 10.1109/TDSC.2022.3216297.

[5] Smriti Bhatt, Farhan Patwa, and Ravi Sandhu. "An Attribute-Based Access Control Extension for OpenStack and Its Enforcement Utilizing the Policy Machine". In: *2016 IEEE 2nd International Conference on Collaboration and Internet Computing (CIC)*. 2016, pp. 37–45. DOI: 10.1109/CIC.2016.019.

[6] Smriti Bhatt, Thanh Kim Pham, Maanak Gupta, James Benson, Jaehong Park, and Ravi Sandhu. "Attribute-Based Access Control for AWS Internet of Things and Secure Industries of the Future". In: *IEEE Access* 9 (2021), pp. 107200–107223. DOI: 10.1109/ACCESS.2021.3101218.

[7] Prosunjit Biswas, Ravi Sandhu, and Ram Krishnan. "Label-Based Access Control: An ABAC Model with Enumerated Authorization Policy". In: ABAC '16. New Orleans, Louisiana, USA: Association for Computing Machinery, 2016, pp. 1–12. ISBN: 9781450340793. DOI: 10.1145/2875491.2875498. URL: https://doi.org/10.1145/2875491.2875498.

[8] William Eiers, Ganesh Sankaran, Albert Li, Emily O'Mahony, Benjamin Prince, and Tevfik Bultan. "Quantifying Permissiveness of Access Control Policies". In: *2022 IEEE/ACM 44th International Conference on Software Engineering (ICSE)*. 2022, pp. 1805–1817. DOI: 10.1145/3510003.3510233.

[9] E. Eugene Schultz. "Risks due to convergence of physical security systems and information technology environments". In: *Information Security Technical Report* 12.2 (2007), pp. 80–84. ISSN: 1363-4127. DOI: https://doi.org/10.1016/j.istr.2007.06.001. URL: https://www.sciencedirect.com/science/article/pii/S1363412707000295.

[10] David Ferraiolo and D. Kuhn. "Role-Based Access Controls". In: *Proceedings of the 15th National Computer Security Conference* (Oct. 1992), pp. 554–563.

[11] B. Fraser. *RFC2196: Site Security Handbook*. USA, 1997.

[12] Ala Al-Fuqaha, Mohsen Guizani, Mehdi Mohammadi, Mohammed Aledhari, and Moussa Ayyash. "Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications". In: *IEEE Communications Surveys & Tutorials* 17.4 (2015), pp. 2347–2376. DOI: 10.1109/COMST.2015.2444095.

[13] Lionel Habib, Mathieu Jaume, and Charles Morisset. "A Formal Comparison of the Bell & LaPadula and RBAC Models". In: *2008 The Fourth International Conference on Information Assurance and Security*. 2008, pp. 3–8. DOI: 10.1109/IAS.2008.18.

[14] Vincent Hu, David Ferraiolo, D. Kuhn, A. Schnitzer, Knox Sandlin, R. Miller, and Karen Scarfone. "Guide to attribute based access control (ABAC) definition and considerations". In: *National Institute of Standards and Technology Special Publication* (Jan. 2014), pp. 162–800.

[15] I. Indu, P.M. Rubesh Anand, and Vidhyacharan Bhaskar. "Identity and access management in cloud environment: Mechanisms and challenges". In: *Engineering Science and Technology, an International Journal* 21.4 (2018), pp. 574–588. ISSN: 2215-0986. DOI: https://doi.org/10.1016/j.jestch.2018.05.010. URL: https://www.sciencedirect.com/science/article/pii/S2215098617316750.

[16] Samuel Jero, Juliana Furgala, Runyu Pan, Phani Kishore Gadepalli, Alexandra Clifford, Bite Ye, Roger Khazan, Bryan C. Ward, Gabriel Parmer, and Richard Skowyra. "Practical Principle of Least Privilege for Secure Embedded Systems". In: *2021 IEEE 27th Real-Time and Embedded Technology and Applications Symposium (RTAS)*. 2021, pp. 1–13. DOI: 10.1109/RTAS52030.2021.00009.

[17] Maria Leitner, Stefanie Rinderle-Ma, and Jurgen Mangler. "AW-RBAC: Access Control in Adaptive Workflow Systems". In: *2011 Sixth International Conference on Availability, Reliability and Security*. 2011, pp. 27–34. DOI: 10.1109/ARES.2011.15.

[18] Ming Mao and Marty Humphrey. "A Performance Study on the VM Startup Time in the Cloud". In: *2012 IEEE Fifth International Conference on Cloud Computing*. 2012, pp. 423–430. DOI: 10.1109/CLOUD.2012.103.

[19] Mehrnoosh Monshizadeh, Vikramajeet Khatri, and Andrei Gurtov. "NFV security considerations for cloud-based mobile virtual network operators". In: *2016 24th International Conference on Software, Telecommunications and Computer Networks (SoftCOM)*. 2016, pp. 1–5. DOI: 10.1109/SOFTCOM.2016.7772161.

[20] Mehrnoosh Monshizadeh, Zheng Yan, Leo Hippeläinen, and Vikramajeet Khatri. "Cloudification and security implications of TaaS". In: *2015 World Symposium on Computer Networks and Information Security (WSCNIS)*. 2015, pp. 1–8. DOI: 10.1109/WSCNIS.2015.7368308.

[21] Jaehong Park and Ravi Sandhu. "The UCONABC Usage Control Model". In: *ACM Trans. Inf. Syst. Secur.* 7.1 (Feb. 2004), pp. 128–174. ISSN: 1094-9224. DOI: 10.1145/984334.984339. URL: https://doi.org/10.1145/984334.984339.

[22] Qasim Mahmood Rajpoot, Christian Damsgaard Jensen, and Ram Krishnan. "Integrating Attributes into Role-Based Access Control". In: *Data and Applications Security and Privacy XXIX*. Ed. by Pierangela Samarati. Cham: Springer International Publishing, 2015, pp. 242–249. ISBN: 978-3-319-20810-7.

[23] Partha Pratim Ray. "An Introduction to Dew Computing: Definition, Concept and Implications". In: *IEEE Access* 6 (2018), pp. 723–737. DOI: 10.1109/ACCESS.2017.2775042.

[24] J.H. Saltzer and M.D. Schroeder. "The protection of information in computer systems". In: *Proceedings of the IEEE* 63.9 (1975), pp. 1278–1308. DOI: 10.1109/PROC.1975.9939.

[25] R.S. Sandhu, E.J. Coyne, H.L. Feinstein, and C.E. Youman. "Role-based access control models". In: *Computer* 29.2 (1996), pp. 38–47. DOI: 10.1109/2.485845.

[26] Amazon Web Services. *Cloud Computing Services - Amazon Web Services (AWS)*. n.d. URL: https://aws.amazon.com/ (visited on 03/28/2023).

[27] Amazon Web Services. *Getting Started - Amazon Simple Storage Service (S3) - AWS*. n.d. URL: https://aws.amazon.com/s3/getting-started/ (visited on 04/06/2023).

[28] Amazon Web Services. *What is Cloud Computing*. n.d. URL: https://aws.amazon.com/what-is-cloud-computing/ (visited on 04/26/2023).

[29] Farhan Bashir Shaikh and Sajjad Haider. "Security threats in cloud computing". In: *2011 International Conference for Internet Technology and Secured Transactions*. 2011, pp. 214–219.

[30] Guoliang Tang, Feng Yang, Zhiyong Zhang, and Jiexin Pu. "An Extended Role-based Access Controls Model: Temporal, Spatial, Workflowed and Attributed Role-based Access Controls Model". In: *2010 Fourth International Conference on Genetic and Evolutionary Computing*. 2010, pp. 534–537. DOI: 10.1109/ICGEC.2010.137.

[31] Xiaopeng TIAN and Haohao SONG. "A zero trust method based on BLP and BIBA model". In: *2021 14th International Symposium on Computational Intelligence and Design (ISCID)*. 2021, pp. 96–100. DOI: 10.1109/ISCID52796.2021.00031.

[32] Mumina Uddin, Shareeful Islam, and Ameer Al-Nemrat. "A Dynamic Access Control Model Using Authorising Workflow and Task-Role-Based Access Control". In: *IEEE Access* 7 (2019), pp. 166676–166689. DOI: 10.1109/ACCESS.2019.2947377.

[33] Claes Wohlin, Per Runeson, Martin Höst, Magnus C. Ohlsson, Björn Regnell, and Anders Wesslén. *Experimentation in Software Engineering*. Springer Publishing Company, Incorporated, 2012. ISBN: 978-3-642-29043-5.

[34] Shucheng Yu, Cong Wang, Kui Ren, and Wenjing Lou. "Achieving Secure, Scalable, and Fine-grained Data Access Control in Cloud Computing". In: *2010 Proceedings IEEE INFOCOM*. 2010, pp. 1–9. DOI: 10.1109/INFCOM.2010.5462174.