AUTHORIZATION FOR WORKLOADS IN A DYNAMICALLY SCALING, HETEROGENEOUS SYSTEM

Pushpalanka Rajaluxmie Jayawardhana

158217G

Degree of Master of Science

Department of Computer Science and Engineering

University of Moratuwa
Sri Lanka

April 2019

AUTHORIZATION FOR WORKLOADS IN A DYNAMICALLY SCALING, HETEROGENEOUS SYSTEM

Pushpalanka Rajaluxmie Jayawardhana

158217G

Thesis submitted in partial fulfillment of the requirements for the degree Master of Science

Department of Computer Science and Engineering

University of Moratuwa
Sri Lanka

April 2019

DECLARATION

I declare that this is my own work and this thesis does not incorporate without acknowledgment any material previously submitted for a Degree or Diploma in any other University or institute of higher learning and to the best of my knowledge and belief, it does not contain any material previously published or written by another person except where the acknowledgment is made in the text.

In addition, I hereby grant to the University of Moratuwa the non-exclusive right to reproduce and distribute my thesis, in whole or in part in print, electronic or other medium. I retain the right to use this content in whole or part in future works.

…………………………………………	………………………….
Pushpalanka Rajaluxmie Jayawardhana	Date

The above candidate has carried out research for the Masters thesis under my supervision.

…………………………………………	………………………….
Prof. Gihan Dias	Date

ABSTRACT

Enterprises in the modern world have gone through a phase of digital transformation which has contributed immensely in the growth of enterprise systems. This has spread through concepts such as e-government, open banking, e-healthcare, e-commerce concepts to digitalized organizations. Conventionally, systems ran within the corporate infrastructure. In the past few years, organizations have been moving to the cloud. Authentication and authorization work well in on-premises or within a single cloud. But authentication and authorization in modern systems with hybrid cloud and multi-cloud approaches where none of the parties individually govern the perimeter of the system is still an open problem. The components serving in one part of the system can be totally strange to the other party and is not aware of the security privileges they have. On the other hand, enterprise systems cannot compromise on information security, though they may want to have the advantages of multi-cloud systems. While there have been several attempts done by the research communities from Google, Docker, Dropbox etc. to provide a common identification protocol across systems, authorization mechanisms still lacks attention. This research provides a solution for authorization between multiple systems (on-premise and cloud or multiple clouds) based on identification completed by the infrastructure. In the provided solution, a central server assigns attested identity to each legitimate workload, to identify them and apply authorization policies at resource access. The resource servers reside behind an access control layer, which allows method execution according to an administrator-defined policy that considers fine-grained details such as the accessing resource, action to be performed and other context details, in addition to the identity of the consumer and the resource.

Keywords: Authorization, Access control, Multi-Cloud, Hybrid Cloud

ACKNOWLEDGMENTS

I would like to express profound gratitude to my supervisor, Prof. Gihan Dias, for his continued supervision, invaluable support, useful suggestions throughout the research work and forewarnings on possible pitfalls. His experience, expertise, and continuous guidance encouraged and enabled me to complete this research work successfully.

I also express my gratitude to Mr. Prabath Siriwardena, my external project supervisor for providing me with the initial project idea, giving invaluable assistance throughout and introducing me to the direct technical community around latest technologies relevant to the project. It was a great identification of an emerging industrial requirement, which I could passionately work on to provide a solution.

I am grateful for all the support given by Dr. Indika Perera, to complete this research project, with smooth handling of all the processes. I am extending my thanks to all the staff from the Department of Computer Science and Engineering for their kindness expressed in all occasions.

I would like to thank WSO2 Lanka (PVT) Ltd, for the encouragement given on proceeding with higher education and sponsorship given to follow the MSc. I am thankful to my colleagues and friends for their continued assistance and encouragement. I would like to especially thank Mr. Chamila Wijayarathna for helping me to get research materials.

I am as ever, especially indebted to my parents for their support throughout my life. Their love and blessings kept me encouraged at the hardest times to complete this project, despite the obstacles.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

ABAC	- Attribute Based Access Control
CI/CD	- Continuous Integration/Continuous Development
CRAM	- Challenge Response Authentication Mechanisms
CNCF	- Cloud Native Computing Foundation
DAC	- Discretionary Access Control
DCR	- Dynamic Client Registration
GDPR	- General Data Protection Regulation
IAM	- Identity and Access Management
IID	- Instance Identity Document
IIT	- Instance Identity Token
MAC	- Mandatory Access Control
MSA	- Micro-Services Architecture
MSI	- Managed Service Identity
OPA	- Open Policy Agent
PAP	- Policy Administration Point
PDP	- Policy Decision Point
PEP	- Policy Enforcement Point
PIP	- Policy Information Point
RBAC	- Role Based Access Control
SaaS	- Software as a Service
SPIFFE	- Secure Production Identity Framework For Everyone
SPIRE	- SPIFFE Runtime Environment
STS	- Security Token Service
SVID	- SPIFFE Identity and Verifiable Identity Document
XACML	- eXtensible Access Control Markup Language

1. INTRODUCTION

The term ‘Cloud’ is nowadays used for various technologies in different contexts but generally refers to computing provided as a service from a remote location. While there is debate on who first introduced the term ‘Cloud Computing’, Ramnath Chellappa, a professor in University of Texas, has defined it as a new “computing paradigm where the boundaries of computing will be determined by economic rationale rather than technical limits alone” in 1997, which several sources treat as the first application of the word in the current context[1]. **In the context of this document, the ‘cloud’ is referred to as the most general use case of a dynamically scaling, heterogeneous system**.

An enterprise has a specific business focus, while digital transformation can support it to better achieve its goals. The more they can delegate the operation and maintenance of digital infrastructure, the more they can focus on the business. With vendors such as Amazon emerging to provide a variety of Cloud services, these enterprises have tried to make use of them with the benefits of less maintenance, low cost due to elastic scaling, low initial cost and less time to go to market. This movement paved the way for CSPs (Cloud Service Providers) and related technologies.

While the ‘Cloud’ provides the above benefits, it also raises concerns on how secure it is and the enterprises have been worried about the control they get while running on third-party infrastructure, reliability, accountability and the privacy of the data and functionality of the system. Research communities have been working on multiple directions to find solutions to these issues, resulting in amazing technologies such as containers, container orchestration, virtualization etc. that broaden the horizons of capabilities of computing, enabling enterprises to widespread the systems. This project is an effort to contribute to this initiative in the access management aspect towards a better-controlled system.

1.1. Multi-cloud Environments

With different vendors and communities getting established in multiple segments of computing and with rapid technological innovations technologies emerging in the computing arena, heterogeneity is inevitable. Large enterprise systems depend on different technology providers to get the best possible service on their requirements in the most cost-effective manner.

With different vendors providing cloud services, enterprise systems now consist of multi-cloud environments, often from several vendors. These may be combined with in-house infrastructure and services. Below is the current major categorization of clouds that can be seen widely.

1. Private cloud - an automated, highly virtualized installation of IT infrastructure managed by an organization’s own IT team.
2. Public cloud - makes use of an infrastructure-as-a-service (IaaS) offering managed by a third-party provider such as AWS, Google Cloud Provider or Microsoft Azure.
3. Hybrid cloud - refers to the combined use of at least one private cloud and at least one public cloud service, with some degree of integration between the two cloud environments.
4. Multi-cloud - indicates the use of more than one public cloud service. Also refers to as community clouds sometimes.

As per a survey[2] done by Nutanix in mid-2018, an emerging competing vendor for AWS, the IT decision makers have shown more interest towards hybrid cloud and predicted the usage will grow in the future, as in below Figure 1.1[2]. It further mentions that 97% of these respondents said that being able to move applications easily between clouds is a requirement. While this interoperability between cloud types is the top benefit that respondents have been looking in the hybrid cloud computing, availability and avoiding vendor lock-in has also been a motive.

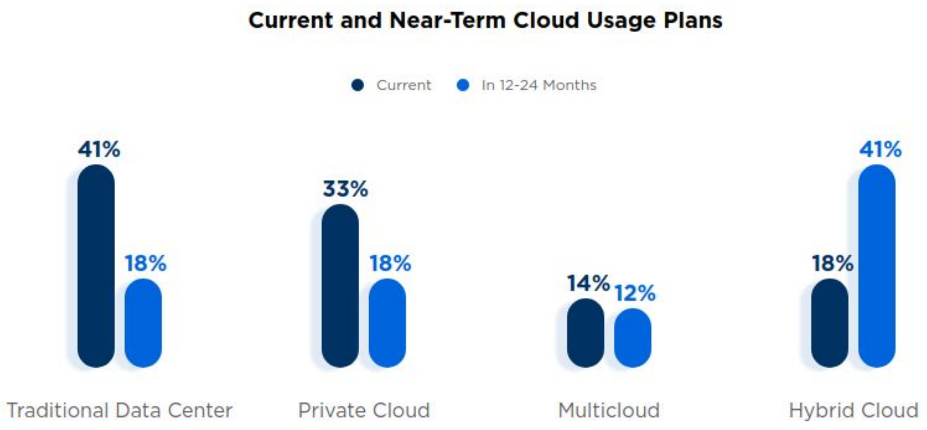


Figure 1.1 - Cloud Usage Plans[2]

As seen above, it is evident that enterprise systems are heading towards hybrid cloud option without depending on one cloud service provider, which raise the requirements on interoperability, including the Identity and Access Management (IAM) aspects of interactions among multiple clouds.

1.2. Problem Statement

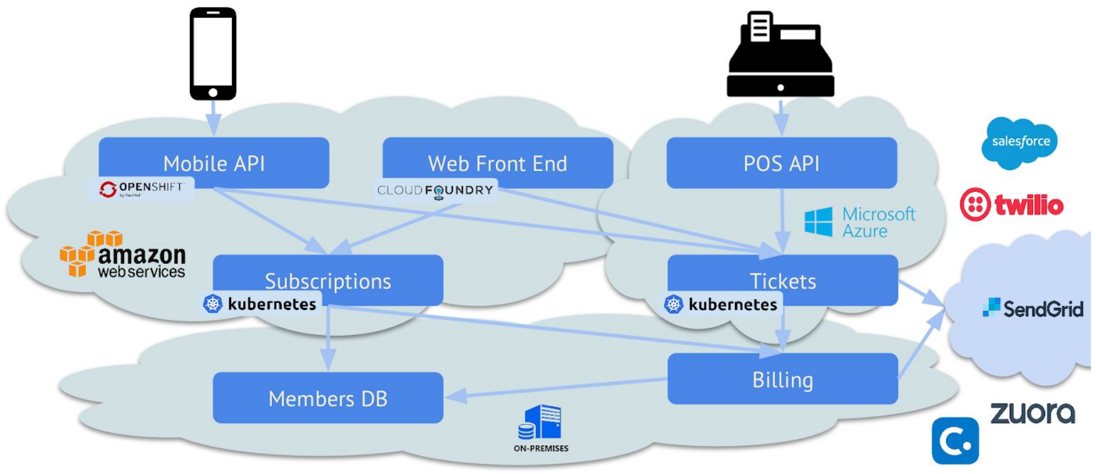


Figure 1.2 - A Multi-cloud Environment Used by an Enterprise System[3]

As per the Gartner report ‘Hype Cycle for Cloud Computing, 2018’[4], Multi-cloud is currently in the peak of the hype cycle, which is exposing its practicality challenges. It also mentions ‘Cloud Security Assessments’ are climbing the slope, which indicates the security challenges of this paradigm.

With:

- new scales of components that deal with each other in the enterprise systems
- the paradigm shift from monolithic applications to microservice-based applications, and
- CI/CD(Continuous Integration/Continuous Development) nature of the systems to cater to changing requirements of the consumers,

establishing trust, maintaining it and then authorizing the actions these components perform on each other has become quite challenging. Proper authorization policies are required and should be made effective, to control the misuses and ensure the intended functionality in a highly distributed system,. To apply these policies, the involved parties need to be proven with identities and these identities should be mapped with the defined policies. Addressing this in a highly distributed system that has zero trusts established requires a proper framework in place. This has also been identified in the Gartner Technical Professional Advice “Building Identity Into Microservices” 2017 as, “MSA (Micro-Services Architecture)-specific IAM (Identity and Access Management) is still in its infancy. The primary focus of the MSA community thus far has been authentication and, more narrowly, the use of OAuth 2.0, leaving other important questions, such as the authorization architecture, unaddressed”[5].

Below challenges can be listed in the current high scale systems,

- Security groups concept used by cloud providers by grouping nodes and applying policies on the group, has become too coarse-grained for microservices based environments.
- IP based ACLs don't work as they cannot guarantee authorization when dynamic IPs are in place. Also even in a kubernetes pod, multiple services

can be running which raise the requirement of a finer grained level of identity and authorization.

- Once this granular identification has happened in a highly dynamic heterogeneous system, binding authorization to these short lived entities is a challenge.

While all of the above challenges are present when running a system on the cloud, more challenges of another scale appear when a system is deployed across multiple clouds. In other words, the system is built of several components running in different cloud systems that work together on a common purpose. The major challenge comes on how to uniquely identify components/processes without any ambiguity, in a trusted manner cross these clouds. As the CSPs work individually in their individual trust domains and with the dynamic nature of the cloud, current technologies struggle in this use case. While the research community has developed technologies to address authentication in this case, this research addresses the authorization aspect which comes next, considering the above challenges.

We, therefore, frame our research question as:

How do we define and implement an authorization architecture for a multi-cloud enterprise system?

1.3. Objectives

This research achieves the following objectives:

- Study existing models, solutions and standards that support authorization within an enterprise system, their advantages, and limitations. Study the cloud environments and their inherent characteristics.
- Study the other relevant aspects of authorization such as authentication and administration of access control policy as required by the authorization architecture for a cloud system.
- Build up the components of the architecture, that can coexist with the current enterprise systems, providing authorization capabilities across clouds, with minimal effort to integrate.

Innovations or implementations on trust bootstrapping in the system is not done as part of the project. Rather than reinventing the wheel, existing implementations that provide trust bootstrapping and authentication for a system with considered characteristics of scale and dynamic nature were evaluated and best-suited technology is used. On top of this authentication solution, the authorization of the overall system is addressed.

1.4. Research Contribution

With the industry interest towards multi-cloud systems, the research communities have come up with standards and approaches to address authentication among multi-cloud systems. The next steps in this journey, with respect to Information System Security, is 'authorization' between these systems. This research is to take steps toward achieving this objective as following,

- Provide a solution for authorization among multi-cloud systems
- Development of the required components in the proposing design and architecture, filling the lacking parts and making use of available components whenever possible.

- Provide a proof of concept demonstration of the designed architecture.
- Convenience analysis on how easily this architecture can be utilized by an existing entity that is moving to a multi-cloud system.

1.5. Outline

The rest of the chapters of this thesis are organized as follows.

Chapter 2 is a thorough look into the related work on access control having a deeper look into the concepts of authentication and authorization. It discusses access control approaches such as DAC, MAC, and RBAC, followed with modern access control approaches of ABAC, XACML, and OPA. This chapter also looks back at the classical security models for access control, such as Bell-La-Padula model and BIBA model addressing confidentiality and integrity and also introduces several other models establishing the foundation of modeling access control. Then this chapter includes a concise look on 'Cloud' as a primary example of a dynamically scaling, heterogeneous system that is trending in the current information technology landscape. With this, the chapter narrows down the focus more on the authentication and authorization technologies specific to workloads that are running in a cloud-like environment and investigates more in this direction. In authentication wise it discusses, credential-based authentication, Needham-Schroeder protocol, Kerberos, SPIFFE standard and OAuth in detail.

Research methodology is presented in Chapter 3. The rationale of technology selection and reasoning behind the architectural decisions are discussed here. This specifically focuses on the authentication technology selection used in the solution and authorization approach.

Chapter 4 presents implementation details specific to the architecture defined in the previous chapter.

Performance evaluation of the implementation is presented in Chapter 5. It evaluates the proposed architecture against it strong points, weak points, it’s best-fit use cases, worst fits, and adaptability for an existing enterprise system.

Chapter 6 discuss the achievements of the system, limitations of the proposed solution and identified future improvements and research work that can be acted upon.

2. LITERATURE REVIEW

2.1. Access Control

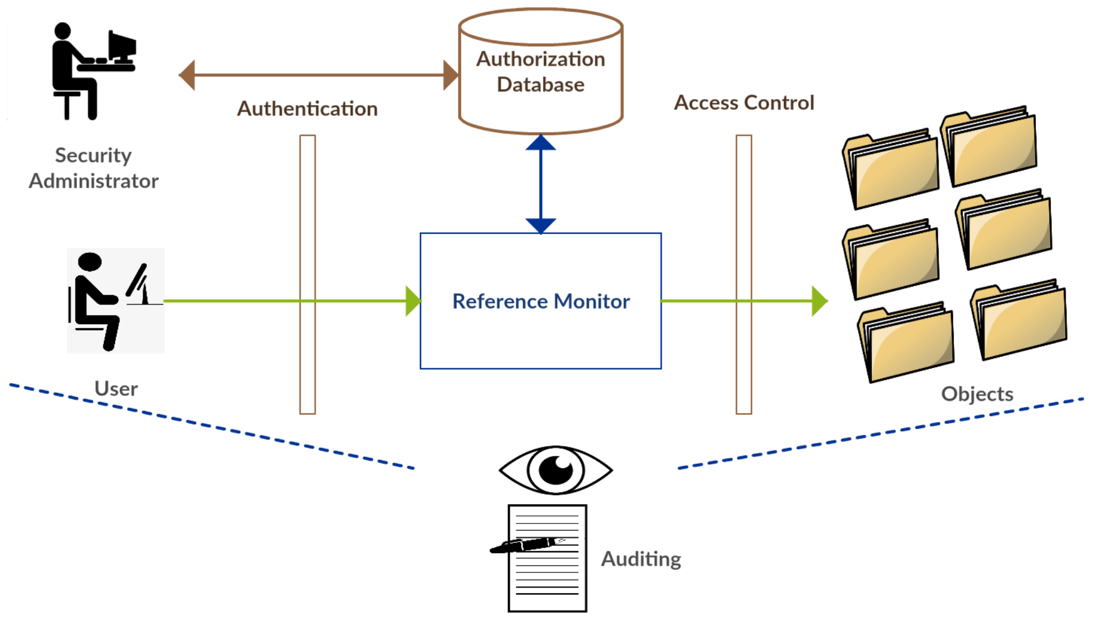


Figure 2.1 - Access Control and Other Security Services[6]

When resources are of value and should not be exposed to or be accessed by everyone, access to these resources needs to be controlled. Authentication, Authorization, and Accounting (AAA) architecture has been used methodically in providing these access control capabilities, mostly in the IP-based networks for tracking user activities on network resources. Additionally, it is considered a generic architecture applicable to other systems too [7].

AAA architecture,

- Authentication
- Authorization
- Accounting

Access control as a whole is related with all the above 3 aspects as seen in Figure 2.1, which a redraw of a diagram from the paper 'Access control: principle and practice' by R. S. Sandhu and P. Samarati. It is about enforcing appropriate

authorization for the system based on the user's identity, serving the objective of protecting the system resources against inappropriate or undesired access.
In the follow-up content, the first 2 aspects of access control are discussed in detail while accounting is treated as a concept that should be applied across.

2.2. Authentication

Authenticity verification is the foundation where other security concepts such as Authorization and Accounting are based on. Identifying an entity such as a person, a group, a device or an application to be what they declare to be is the challenge addressed by authentication. In other words, authentication is to bind a subject to an identity that uniquely identifies the respective subject from others. The basis of identification is majorly based on below 3 factors as below.

- Something Known

This uses a factor that can be known only to the entity that is to be identified.

Eg: passwords, PIN numbers, secret keys

- Something Possessed

This is mostly based on a physical accessory that is present with the entity to be authenticated.

Eg: electronic token generators, ship/smart card

- Something Inherent

In the scope of a person, this is mostly the biometric features and involuntary actions.

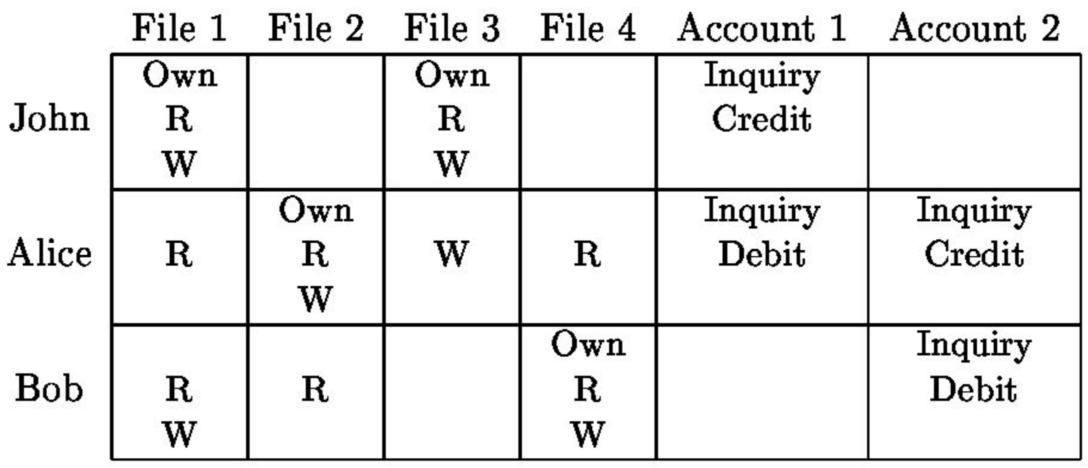
Eg: a handwritten signature, fingerprint, voice, retina

Based on the asset value of the resource, modern enterprise systems use a combination of these factors rather than depending on one factor, for enhanced security, providing harder challenges against impersonation. This is known as Multi-factor authentication(MFA) in the domain.

2.3. Authorization

Once the subject identity is authenticated, the system needs to figure out the actions it can perform on the resources, before the entity is allowed to act. Authorization decides what actions subject may or may not do within the system. The resources are protected by an access control layer, which will check for the authorization level of the entity and allowed resources as a part of this.

2.3.1. Access Control Matrix



	File 1	File 2	File 3	File 4	Account 1	Account 2
John	Own R W		Own R W		Inquiry Credit	
Alice	R	Own R W	W	R	Inquiry Debit	Inquiry Credit
Bob	R W	R		Own R W		Inquiry Debit

Figure 2.2 - Access Matrix [6]

Above figure 2.2, shows a sample access matrix that defines allowed access levels for three subjects, on 4 files and 2 accounts, which is another redraw of an image from the paper by R. S. Sandhu and P. Samarati. Here the R denotes Read and W denotes Write access. Whenever requests from the subjects come to access one of these files or accounts, this access matrix should be referred to determine the allowed access level and decide whether to deny the request or allow it.

As seen from the above matrix this can become complex when the number of resources, number of subjects and access levels on each resource grows. In order to address high number of subjects in the matrix, on some occasions, these subjects are assigned to groups and these groups are used in the access control matrix in the place of subjects.

2.3.1.1. Access Control List

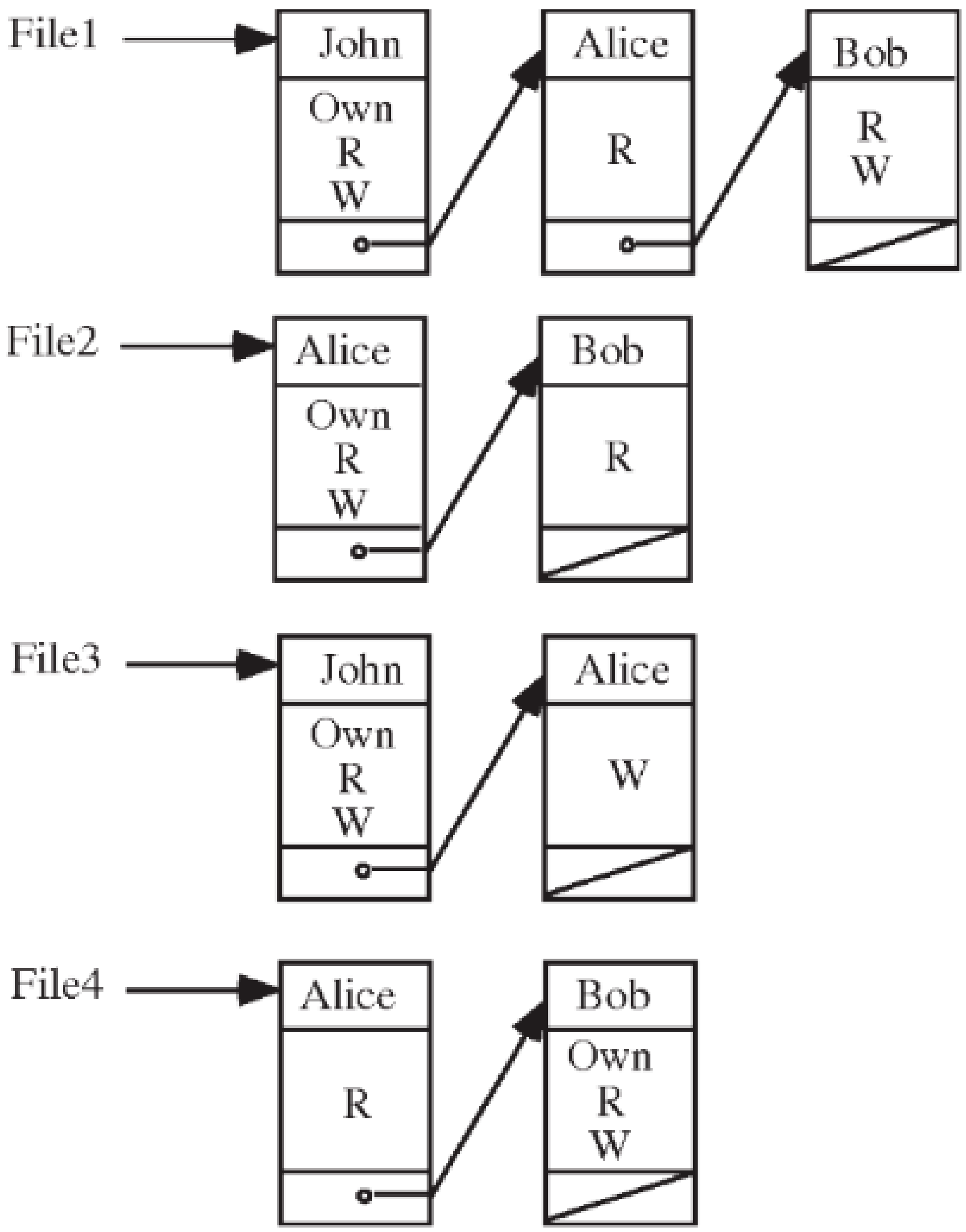


Figure 2.3 - Access Control List for Files[6]

Access Control Lists is a way of implementing the access matrix, representing access levels allowed on an object for subjects as seen in above figure 2.3. As shown in the list, in this implementation object is used as the key and subjects and their access levels are tracked against them. This has another flavor named 'Capabilities List' where the subject is used as the key and objects they can access is tracked along with the access levels. One of the lists is used based on the most frequent lookup happen in the system, either from object or subject.

2.3.2. Discretionary Access Control (DAC)

In this mode of access control, a request to access a resource is granted if it is authorized and rejected otherwise. DAC policies control access to the resource by an entity(subject). Hence once the entity gets access to the resource, they can pass on the resource to another entity and that information flow is not governed by DAC, which is a disadvantage. This is also called 'the safety problem' of propagation of access rights. Additionally, access privileges for objects are decided by the owner of the object in DAC, rather than a global policy or administrator enforcing them, which can be complex without central control.

2.3.3. Mandatory Access Control (MAC)

MAC policies are defined using security labels attached to the resources(objects) that are to be accessed by the subjects. MAC-based systems are governed by policy administrators who implement security policies enforced on all the users of the system. Rest of the users are not allowed to modify or override these policies deliberately or accidentally. MAC is used when the risk of attack is very high while confidentiality is a primary access control concern in military and intellectual contexts.

2.3.4. Role Based Access Control (RBAC)

When modern enterprise systems are concerned, there can be millions of subjects accessing the system. Even though the above two DAC and MAC policies are recognized by the Orange Book of the US Department of Defense as well for their strength in specific domains, these policies fall short to cater requirements of many of the modern commercial and government systems, with the scales they deal with. As a solution to this, RBAC has been introduced, where subjects are assigned a role based on the security clearance levels each of them can have. Then the access policy is defined based on these groups.

Eg: A role named ‘lecturer’ will be defined and assigned to all the lecturers in a university. Then a mark sheet resource’s access level will be governed as any subject with the role ‘lecturer’ can write and read it while none other can write to it.

An administrator should additionally govern the role assignment to users to define the group, apart from the policy governance, in this approach.

2.3.5. Attribute-Based Access Control (ABAC)

Taking forward the RBAC approach one more step, ABAC approach is defined. This approach generalizes the RBAC control, such that it can be applied to other attributes as well. These attributes can be subject-related factors like role, age, location etc, objects related attributes such as classifications, size, age or environment-related attributes such as time. This is also referred to as ‘fine-grained access control’ due to the detail-oriented flexibility provided in the approach. OASIS Extensible Access Control Markup Language(XACML)[8] is an open standard that defines this approach in a formal manner. This standard is discussed below.

2.3.5.1. XACML

This open standard ‘Extensible Access Control Markup Language’ has evolved through a considerable time from its 1.0 version in 2013 to 3.0 version[8] in 2017. It defines the format to write policies to cater for ABAC requirements. It has a set of standards to define the interoperability between the components required in the ecosystem.

Policy Administration Point (PAP) - This component is used to define access control policies.

Policy Decision Point (PDP) - This is the policy decision taking engine, which runs the incoming authorization request against the defined policies and commands whether to allow or deny.

Policy Enforcement Point (PEP) - This acts as the gate that secures the resources. When an access request comes in, this component requests the decision on that from the PDP and act upon the decision, allowing the request to access the resource or denying.

Policy Information Point (PIP) - This is a supportive component to the PDP. At the policy execution time, if there are additional attributes required by the PDP on the environment, subject or object, PIP is responsible to retrieve it in a trusted manner.

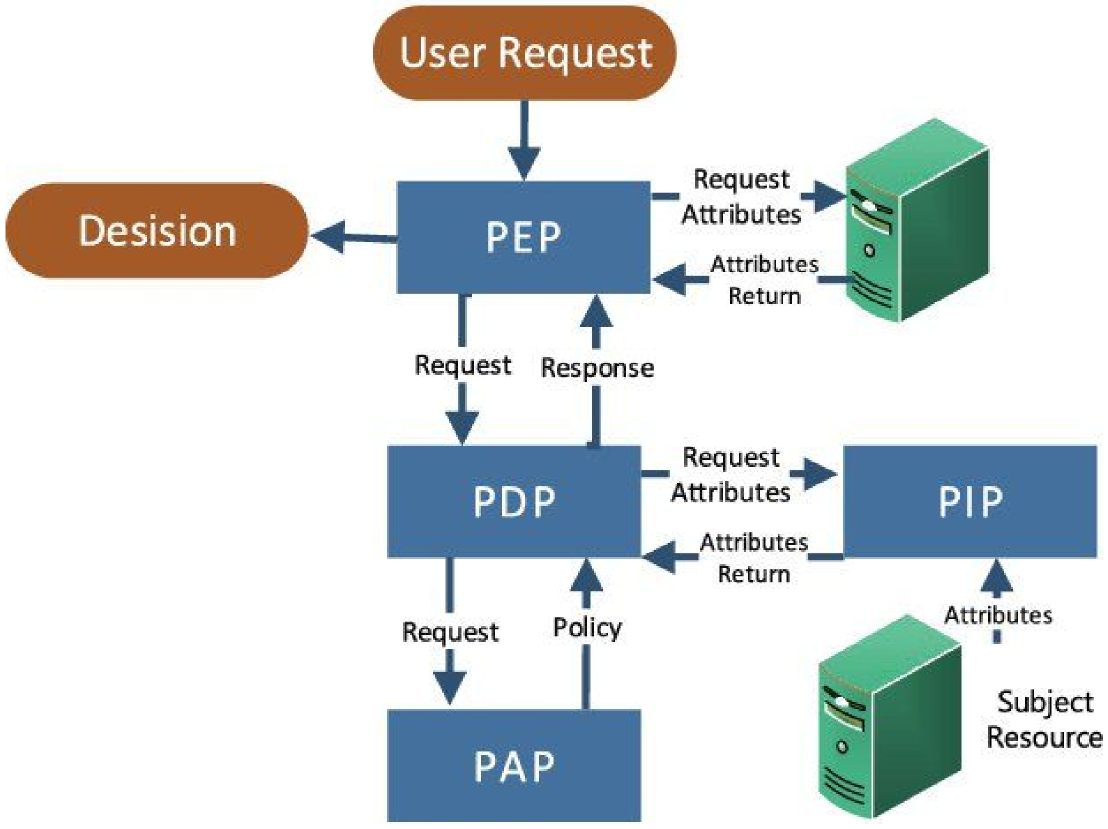


Figure 2.4 - XACML Based Access Control Components

A sample XACML policy can be found in Appendix A, which states a policy as below in plaintext.

- Anyone who is trying to access the resource 'foo//*' should be authorized under this policy.
- If 'admin' subject is trying to read, write or delete, allow it.
- If anyone is trying to 'read' the resource 'foo/abc', between the time '09:00:00+05:00 GMT and 16:00:00+05:00 GMT' allow it only if the 'subject's email address is ending with 'abc.com'.

XACML has recently introduced a specification making it more REST (REpresentational State Transfer) friendly with accepting JSON requests and JSON response in addition to XML format[9]. However, there is no specification as of now to define XACML policy in JSON format, though individuals have put an effort in that direction. The main reason for this seems XACML policies stay at rest in the engine and only requests and response needs to be transferred through the wire in an optimal way. XACML adaptation in the industry has been challenged due to it's involved complexities in writing the policies.

2.3.5.2. OPA

Open Policy Agent[10] is a Policy-based access control solution for cloud-native environments which is accepted by CNCF (Cloud Native Computing Foundation).

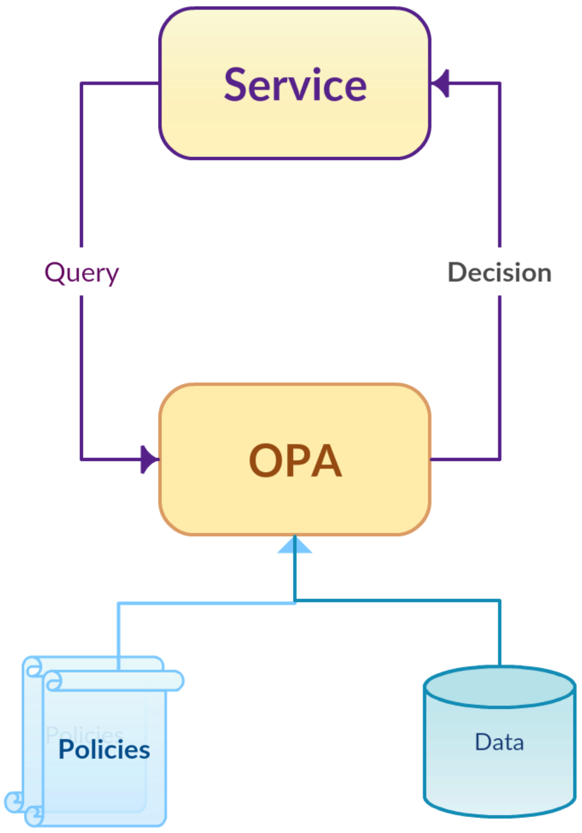


Figure 2.5 - How OPA works

This agent depends on a data set that is injected to it's engine, similar to below.

```
subordinates = {"alice": [], "charlie": [], "bob": ["alice"], "betty":
["charlie"]}
```

```
# HTTP API request
import input as http_api
# http_api = {
#   "path": ["finance", "salary", "alice"],
#   "user": "alice",
#   "method": "GET"
#   "user_agent": "cURL/1.0"
#   "remote_addr": "127.0.0.1"
# }


default allow = false

# Allow users to get their own salaries.

allow {
  http_api.method = "GET"
  http_api.path = ["finance", "salary", username]
  username = http_api.user
}


# Allow managers to get their subordinates' salaries.
allow {
  http_api.method = "GET"
  http_api.path = ["finance", "salary", username]
  subordinates[http_api.user][_] = username
}


# Allow managers to edit their subordinates' salaries only if the request came
from user agent cURL and address 127.0.0.1.

allow {
  http_api.method = "POST"
  http_api.path = ["finance", "salary", username]
  subordinates[http_api.user][_] = username
  http_api.remote_addr = "127.0.0.1"
  http_api.user_agent = "curl/7.47.0"
}
```

In the above simple policy it defines a concept named subordinates and define that relationship between the entities. Then it defines the rules that give the authorization decision as ‘allow’.

OPA also provides guidance on how to use this technology to satisfy requirements that have been addressed by other authorization mechanisms such as RBAC, RBAC with separation of duty, ABAC, AWS IAM and XACML[11]. OPA shows that it is flexible and powerful enough to support the use cases addressed by each of these technologies, by providing approaches to achieve the same functionality. They also provide a REST-based API model to execute and administrate the policies.

2.4. Lattice Based Access Control - Classical Information Security Models

In the early 1980s, the Department of Defence(DoD) of the USA was concerned about the confidentiality of classified military information on computers which were shared by multiple users. This paved the path to the well-known rainbow series which is a set of security standards defining the security handling of computer systems, mainly via the operating system. The “Orange Book” (Trusted Computer System Evaluation Criteria) is the very first of this rainbow series which documented the mechanisms to enforce the confidentiality of data. Common Criteria for Information Technology Security Evaluation superseded this later. This effort has included several security models and has inspired several other models. Below is a brief look into these security models in detail, based on the book ‘Computer Security’ by Dieter Gollmann[12, pp. 115–164].

These security models state a formal policy that defines the criteria to be met to access a resource, by defining a set of controlling rules. The objective of these models is to enforce below notions of security on the system.

Classical Notions of Security

Confidentiality - only the authorized parties access the resource
Unlinkability - unable to state a relation between two observed items of the system
Anonymity - subject is not uniquely characterized within the anonymity set
Integrity - no unintended modifications happened on the resource

Extended Notions of Security

Availability - uninterrupted service

Accountability - the presence of responsibility

Non-repudiation - assurance on undeniable validity

Reliability - assurance on the correctness

From the above notions, in the modern systems put a major focus on the **CIA (Confidentiality Integrity Availability) triad**. It can be assumed that with the distributed nature of the systems and drastic damages caused by attacks such as Distributed Denial of Service (DDoS) on enterprise systems, 'availability' aspect has also gained much attention becoming very critical in the modern systems.

2.4.1. Bell-La-Padula Model

This model uses DAC along with MAC to enforce information policies to preserve confidentiality. It has a two-step approach for access control.

1. Get authorized against a DAC matrix, whose contents can be modified by the subjects.
2. In order to carry out an operation, it should be allowed under the MAC policy, which is not under the control of the subjects.

It defines two properties to be honored in information flow as below.

$\lambda(s)$ - security clearance of the subject

$\lambda(o)$ - security classification of the object

'**Tranquility**' is assumed on these that, once assigned these security labels will remain unchanged(unless modified by a security officer in a secured manner) on the subjects and objects.

Simple Security Property

Subject s can read object o only if,

$$\lambda(s) >= \lambda(o)$$

Star Property

Subject s can write object o only if,

$$\lambda(s) <= \lambda(o)$$

In brief, these two properties state that subjects can read down and write up on objects. One concern on the 'Star property' is that classified objects such as data can be contaminated, damaged or destroyed by an unclassified subject as write up is allowed. Hence sometimes this property is modified as below, so that, write is allowed only on own level, but not up[13].

$$\lambda(s) = \lambda(o)$$

However, there are information flows that can happen under this model, through 'covert channels', opposed to the intended flows. Indirect communication methods such as extracting information from an error message are referred to as a covert channel.

Eg: A database system allows to place data into a database. Direct access is the database is properly controlled via MAC under BLP model. However, in the database system, the time required to place an entry into a database is highly dependent on the current total size of the database. Therefore even though the data sender not allowed to access the total size of the database, the sender can learn this fact based on response time it takes to place the entry.

This is concern made out scope from the BLP model and other engineering practices at design and implementation level needs to address those[13]. Management of these access control policies is also not addressed in the BLP model.

2.4.2. BIBA Model

This model mainly focuses on integrity policies. This model defines below,

Subject : S

Object: O

Integrity Levels captured on a lattice : L

Functions to assign integrity levels : f

$f_s : S \rightarrow L$

$f_o : O \rightarrow L$

Information is allowed to flow downwards only in this model so that subjects with lower integrity levels cannot contaminate the higher integrity level objects. It defines policies to preserve this in different integrity level behaviors of static and dynamic as below.

Static Integrity Levels

Below two properties should hold to prevent high integrity subjects and objects from getting contaminated by low integrity information to preserve its integrity at a static level.

Simple Integrity Property

If s can modify o, then $f_s(s) >= f_o(o) \rightarrow$ no write-up

Integrity Star Property

If s can read o, then s can have write access to some other object p only if,

$f_o(p) <= f_o(o)$

These two properties are the dual for integrity similar to the two properties to preserve confidentiality under the BLP model.

Dynamic Integrity Levels

This defines,

Integrity level **inf**$(f_s(s),f_o(o)) \rightarrow$ the greatest lower bound of $f_s(s)$ and $f_o(o)$

Subject Low Watermark Policy

s can read o at any integrity level. The new integrity level of s is **inf**$(f_s(s),f_o(o))$.

Object Low Watermark Policy

s can modify o at any integrity level. The new integrity level of o is **inf**$(f_s(s),f_o(o))$.

These two policies lower the integrity levels of subjects and objects based on the objects and subjects they interact with. As the integrity levels are only lowered under these policies, there is a risk of all the subjects and objects eventually reach the lowest integrity level and be stuck there.

Policies for Invocation

An extension is possible on this model to address the use case of a subject invoking another subject to access an object, similar to how a subject would access an object via a software tool. This is governed by below property.

Invoke Property

Subject S_1 can invoke subject S_2 only if $f_s(s_2) <= f_s(s_1)$

In the case of S_2 is a software tool that controls this invocation, even a low integrity subject might be allowed to access a high integrity object. In such occasions, the tool performs several checks on the consistency of the object, to make sure it preserves its integrity.

Ring Property

A subject S_1 can read objects at all the integrity levels. It can only modify objects o with $f_o(o) <= f_s(s)$ and it can invoke a subject s_2 only if $f_s(s_1) <= f_s(s_2)$

As seen from above, the invoke property and ring property cannot stand in the same environment. Hence one of these properties should be selected and applied as appropriate in the use case.

2.4.3. Chinese Wall Model

Brewer and Nash (1989) proposed a policy called the ‘Chinese Wall Policy’ that addresses the conflicts of interest, been inspired by the Clark and Wilson’s paper[14] to pay attention to commercial security needs, at a time when military security thinking has been dominating the computer security arena[15]. This model makes more sense in business domains such as investment banking and consulting. If the same analyst is been consulted by two competing companies, it gives rise to a conflict of interest situation, as the same person has access to sensitive details of two competitors. Hence, a set of policies are introduced to address this. This is not an integrity policy, but an access control confidentiality policy, that looks into a more specific commercial use case than the BLP model.

The security policy builds on three levels of abstraction.

O - Objects such as files. Objects contain information about only one company.

C - Company groups.

P(C) - Conflict classes cluster. The groups of objects for competing companies.

> Eg.
>
> conflict classes:
>
> {Toyota, Honda, BMW, Mercedes Benz, Ford}
>
> {Apple, SAMSUNG, Huawei, Motorola}

Simple Security Property

A subject 's' can be granted access to an object 'o' only if the object:

- is in the same company datasets as the objects already accessed by s, that is, "within the Wall," or belongs to an entirely different conflict of interest class.

Chinese Wall *-property

Write access is only permitted if:

- access is permitted by the simple security rule
- no object can be read,
    - which is in a different company dataset than the one for which write access is requested
    - contains unsanitized information.

2.4.4. Clark-Wilson Model

Clark and Wilson suggest a model focusing on commercial systems, in the paper[14] to control confidential information. While this is important in both the commercial and military systems, ensuring the integrity of data to prevent fraud and errors is the main goal in commercial systems. Even if permitted to modify, a user should not be able to corrupt or delete accounting records and logs of the company.

User **authentication** is a critical part of enforcing integrity in a system. From there onwards, there exist two main mechanisms of fraud and error control from the pre-computing era, namely the well-formed transactions, and separation of duty among employees. Clark and Wilson's model formally defines this as applicable in a computer system.

1. **Separation of Duties** - A task is divided to multiple small tasks carried out by two or more different people. This minimizes the chances of a single person acting alone in a fraudulent manner.
2. **Well-formed Transactions** - Users are allowed to perform only a set of well-defined set of legitimate actions on the data. This may be enforced by a tool so that users can't arbitrarily modify the data violating integrity.

Below are rules defined by the Clark-Wilson model[14].

CDI - Constrained Data Items, integrity policy is applied.
UDI - Unconstrained Data Items, not covered under the integrity policy.
IVP - Integrity Verification Procedure, verifies the data items are in a valid state.
TP - Transformation Procedure, transform the data items from one valid state to another.

Certification Rules - Integrity Monitoring

C1 (IVP Certification) - The system will have an IVP for validating the integrity of any CDI.
C2 (Validity) - The application of a TP to any CDI must maintain the integrity of that CDI. CDIs must be certified to ensure that they result in a valid CDI

C3 - A CDI can only be changed by a TP. TPs must be certified to ensure they implement the principles of separation of duties & least privilege

C4 (Journal Certification) - TPs must be certified to ensure that their actions are logged

C5 - TPs which act on UDIs must be certified to ensure that they result in a valid CDI

Enforcement Rules - Integrity Preserving

E1 (Enforcement of Validity) - Only certified TPs can operate on CDIs.

E2 (Enforcement of Separation of Duty) - Users must only access CDIs through TPs for which they are authorized.

E3 (User Identity) - The system must authenticate the identity of each user attempting to execute a TP.

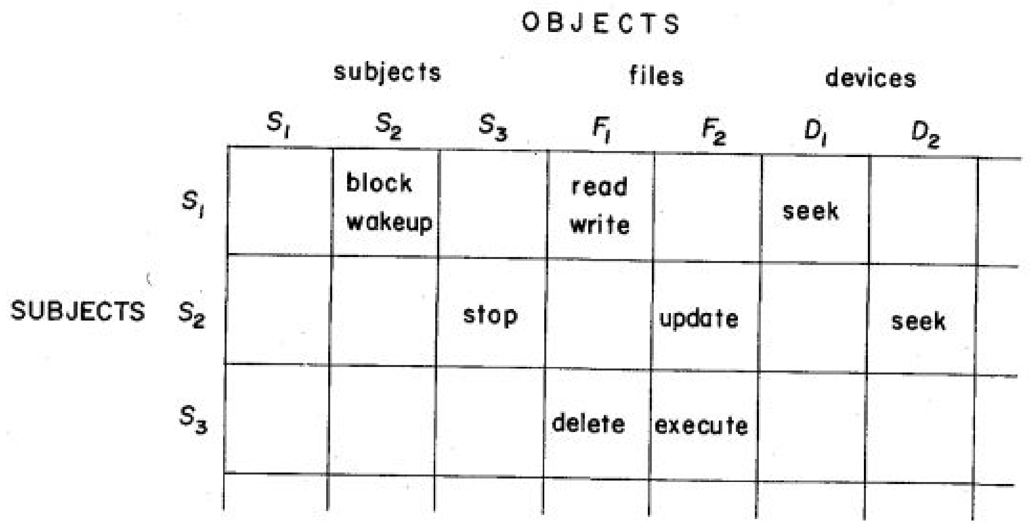
E4 (Initiation) - Only the administrator can specify TP authorizations.

2.4.5. Graham-Denning Model

Introduced in 1972 in a paper 'Protection-Principles and practice' which defines a formal theory to address below 8 concerns.

- How to securely create an object.
- How to securely create a subject.
- How to securely delete an object.
- How to securely delete a subject.
- How to securely provide the read access right.
- How to securely provide the grant access right.
- How to securely provide the delete access right.
- How to securely provide the transfer access right.

It makes use of an access control matrix A, which defines the access rights R on each object and subject. The specialty here is, even the subjects have a designated subject as the controller. It defines the pre-condition for each of these operations, it's impact on the access control matrix A and state transition procedures.



		OBJECTS						
		subjects			files		devices	
		S_1	S_2	S_3	F_1	F_2	D_1	D_2
SUBJECTS	S_1		block wakeup		read write		seek	
	S_2			stop		update		seek
	S_3				delete	execute		

Figure 2.6 - Access Control Matrix of Graham-Denning Model

2.4.6. Harrizon-Ruzzo-Ullman Model

Known as HRU model, introduced in paper 'Protection in operating systems' in 1976, this defines a set of policies for changing access rights and creation and deletion of both objects and subjects inspired by Graham-Denning(GD) model. Extending the GD model, it defines a finite set of procedures that govern modification of access rights of a subject on an object.

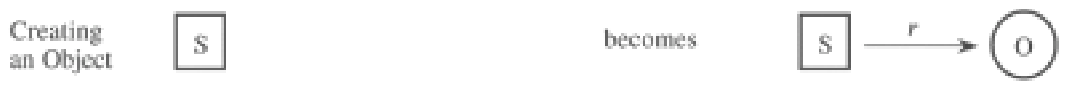
- How to grant right to subject and object pair
- How to delete right from subject and object pair
- How to create subject
- How to create object
- How to delete subject
- How to delete object

Similar to the GD model, this also defines the preconditions and intended state transitions at each of these actions.

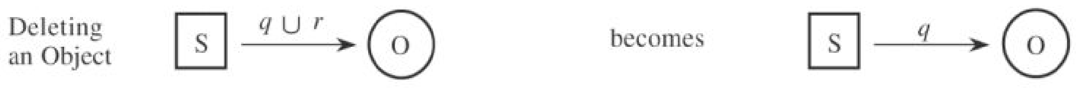
2.4.7. Take-Grant Model

The Take-Grant model is a confidentiality based model that was introduced by Lipton and Snyderin 1977 in a paper “A Linear Time Algorithm for Deciding Subject Security”. It uses a directed graph to indicate the rights passed from one subject to another or from a subject to an object. This model defines four primitive operations as below.

- **Create rule** allows a subject to create new objects

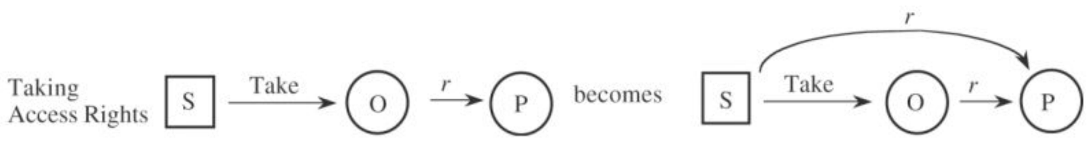


- **Revoke rule** allows a subject to remove rights it has over on another object

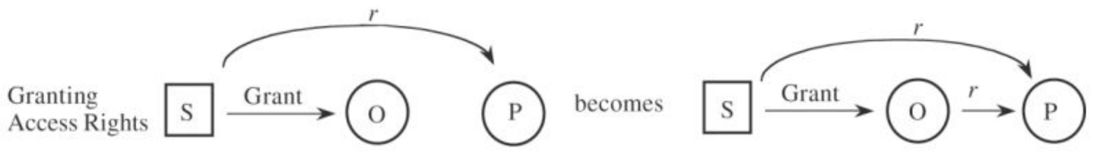


q is no longer in the ‘rights’ set.

- **Take rule** allows a subject to take rights of another object (P)



- **Grant rule** allows a subject to grant ownership rights to another object



With these rules, there are a finite set of states the system can be at a moment, which makes it convenient in evaluating the overall system security.

2.5. Cloud Computing

2.5.1. History

From the initiation of Advanced Research Project Administration Network (ARPANET) which was established in 1969 by the United States government, the computing industry has come a long way, until today's enterprise systems that are running in multi-cloud environments. Below is a brief look back on the journey, before taking a step forward.

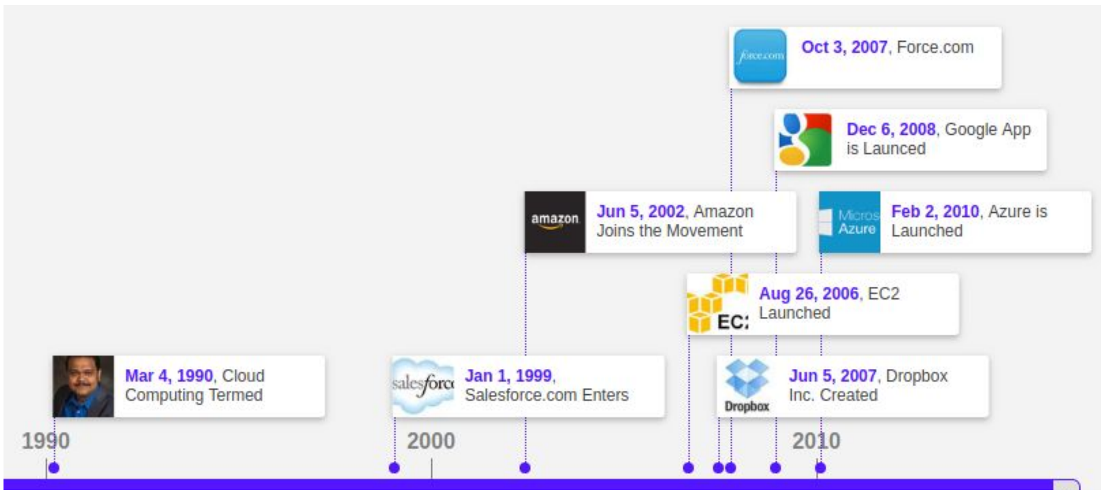

Oct 3, 2007, Force.com
Dec 6, 2008, Google App is Launced
Jun 5, 2002, Amazon Joins the Movement
Feb 2, 2010, Azure is Launched
Aug 26, 2006, EC2 Launched
Mar 4, 1990, Cloud Computing Termed
Jan 1, 1999, Salesforce.com Enters
Jun 5, 2007, Dropbox Inc. Created
1990
2000
2010

Figure 2.7 - History of Cloud Computing[16]

Similar to "Time-sharing" concepts introduced in the 1950s to share the CPU time, to get the most out of the expensive mainframe computers, cloud computing can be thought of as being infrastructure-sharing or platform-sharing approach for modern enterprises. While virtual machines concept allowed multiple systems to run on the same physical platform the movements such as 'Salesforce.com' paved the way to cloud computing with SaaS(Software as a Service) applications followed by PaaS(Platform as a Service), IaaS(Infrastructure as a Service) concepts to share same resources between multiple enterprise systems.

2.5.2. Cloud Services

The enterprise system nowadays has several vendors providing cloud services while having the flexibility to build systems using SaaS, PaaS and IaaS services. Amazon Elastic Compute Cloud (EC2), Google Cloud Provider (GCP), Microsoft Azure can be named as the giants in the industry mainly serving enterprise clouds, but there are a lot of vendors providing different services related with cloud computing such as cloud security, cloud monitoring, cloud orchestration etc. and other cloud storage kind of services such as 'Dropbox', opening new dimensions of the cloud technologies.

The analysts firm Gartner projects public cloud services to be a $206.2 Billion market in 2019 with the continuous predicted growth in the domain, having more focus on SaaS[17].

2.5.3. Hyper-Converged Cloud

With more and more enterprises moving to the cloud or showing interest in the cloud innovations in the domain is obviously required. One of the major concerns the enterprises have in adopting cloud is data security. Having to store sensitive details of their customers, business assets and employees, in a shared infrastructure governed by a third party is a major concern in moving to the cloud. While the hybrid cloud has been addressing this to an extent letting part of the system to run on a private cloud or on-premise, the hyper-converged cloud seems to take the movement forward.

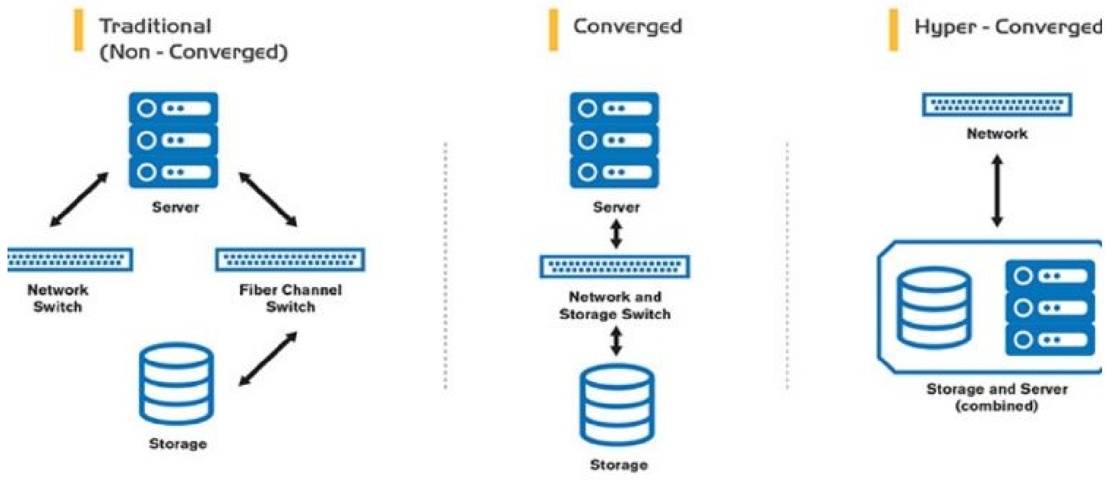


Figure 2.8 - Hyper-Converged Cloud

Hyper-Converged Infrastructure(HCI) facilitates the hyper-converged cloud which is a solution that combines several infrastructure level resources such as servers, storage, and networking as seen in above figure 2.8[18]]. In the context of HCI, a hypervisor provides a virtualized SAN(Storage Area Network) referred to as software-defined storage and virtualized networking referred to as software-defined networking. It facilitates virtualization on top of the storage and computing power for easy management of workloads running in the system. Software tools and commoditizing the underlying infrastructure has made HCI a trend with the rising interest on hybrid cloud, as a unified experience will be supported and making the private clouds or on-premise systems to public cloud abstraction level is made easy. HPE(Hewlett Packard Enterprise) SimpliVity, Nutanix, Microsoft, VMware, and many more vendors compete in this space[19].

2.6. Workloads

2.6.1. What is a workload?

A highly cohesive and de-coupled capability or a unit of work that collectively builds up an enterprise application, which can be running on cloud or on-premise. This can be a regular un-orchestrated process running on a VM or a container scheduled by a container orchestrator[20].

Eg: a microservice, a Kubernetes pod, or a process will be treated as this basic unit as per the deployment. This can be a service provided by a serverless stack similar to Amazon lambda functions as well.

2.7. Workload Authentication Technologies

In order to authorize the entities, proper identification via an authentication process is required. Though these authentication principles stay similar for both humans and workloads, how these entities will prove their identity and the involved verification process needs to be tailored as per the entities. For Workload authentication below technologies are currently available in the information security domain.

2.7.1. Challenge Response Authentication Mechanisms

Shared secret based identification is a basic model that falls under this category, though there are more challenge-response mechanisms nowadays. From the stories of Alibaba and 40 thieves, where a password challenge was submitted to open the cave door, the industry has come a long way make the challenges harder, so that an imposter will not pass. Multi-factor authentication mechanisms which involve biometric challenges and hardware possession based challenges such as SMS OTP(One Time Password), hardware authentication device PIN can be considered as challenges on the possession.

When considering the authentication of workloads, biometric-based authentication might not make sense similar to humans. Yet, considering other unique attributes such as workload initiator, image ID, md5sum or some other hash value comparison of the content of a workload, and providing attribute-based authentication can be considered an option.

2.7.1.1. Username and password based authentication

This is a mechanism that can be used by both humans and systems equally. Here the accessing party is challenged to submit a pre-shared set of credentials, that is expected only to be known to that party and the accused party. Non-repudiation property is not secured in this model.

Eg. When a service is accessing a database system, it will submit a username, password combination shared to it and stored in the database system.

Secure distribution of these secrets and periodic rotation of these secret are major concerns in this model.

2.7.2. Needham–Schroeder protocol

This is protocol suggested in 1978, in the paper 'Using Encryption for Authentication in Large Networks of Computers'[21]. This suggests an approach based on a symmetric key and another approach based on public key cryptography to authenticate two parties and build a secure channel between them for communication, which does not necessarily needs to be between humans. It makes use of a third party(Authentication Server) trusted by both the involved parties to identify and establish the trust which then paved the way to Kerberos protocol. Below is the symmetric key protocol in brief.

A - Alice (Alice want to identify Bob and securely communicate)
B - Bob
AS - Authorization Server
I_{A1} - Nonce generated by A for the transaction
K^{AS} - is a symmetric key known only to A and S
K^{BS} - is a symmetric key known only to B and S
CK - is a symmetric, generated key, which will be the session key of the session between A and B

1. $A \rightarrow AS$: A, B, I_{A1}

A communicated with AS sending own identity, B's identity which it likes to build a communication channel and the nonce generated for the transaction.

2. $AS \rightarrow A$: $\{I_{A1}, B, CK, (CK, A)K^{BS}\}\ K^{AS}$

Once AS received the request from A, it looks up its secret storage, identify the keys of both parties A, B and computes a new session key CK to be used for the conversation. CK is delivered to A, in a secured manner as above. The whole bundle of data is encrypted with A's secret key so that only A can decrypt it. Once decrypted, A can validate the presence of sent nonce, the intended recipient and session key. The importance of recipient's name is to make sure an intruder can't impersonate to be B. This part (CK, A)K^{BS} is encrypted with B's secret key, which means it can only be decrypted by B. Hence A send it as it is to B.

3. $A \rightarrow B$: (CK, A)K^{BS}

Even if another party intercept this message they can't derive the CK, the session key. Hence A and B and securely communicate afterward encrypting messages with

CK, safely assuming the identity of the other party they are communicating with is A or B.

One assumption they have made in this protocol as follows, “We present protocols for decentralized authentication in such a network that are **integrated with the allied subject of naming.**”. This assumption is not true for the use case under consideration in this research, as a naming convention can differ between heterogeneous systems.

2.7.3. Kerberos protocol

The protocol has been invented based on the previously discussed ‘Needham–Schroeder protocol’, to secure network services of the historic project ‘Athena’ distributed system by MIT and IBM. As an alternative to the password, it depends on a Key Distribution Center trusted by the two parties under consideration, to securely introduce each other. Kerberos V4 is the first published version and currently, the protocol is on V5[22], integrating improvements to security and identification related weaknesses in V4. The RFC(Request for Comments) draft by IETF(International Engineering Task Force) introduce Kerberos as below,

> “Kerberos provides a means of verifying the identities of principals, (e.g., a workstation user or a network server) on an open (unprotected) network. This is accomplished without relying on assertions by the host operating system, without basing trust on host addresses, without requiring physical security of all the hosts on the network, and under the assumption that packets traveling along the network can be read, modified, and inserted at will. ”

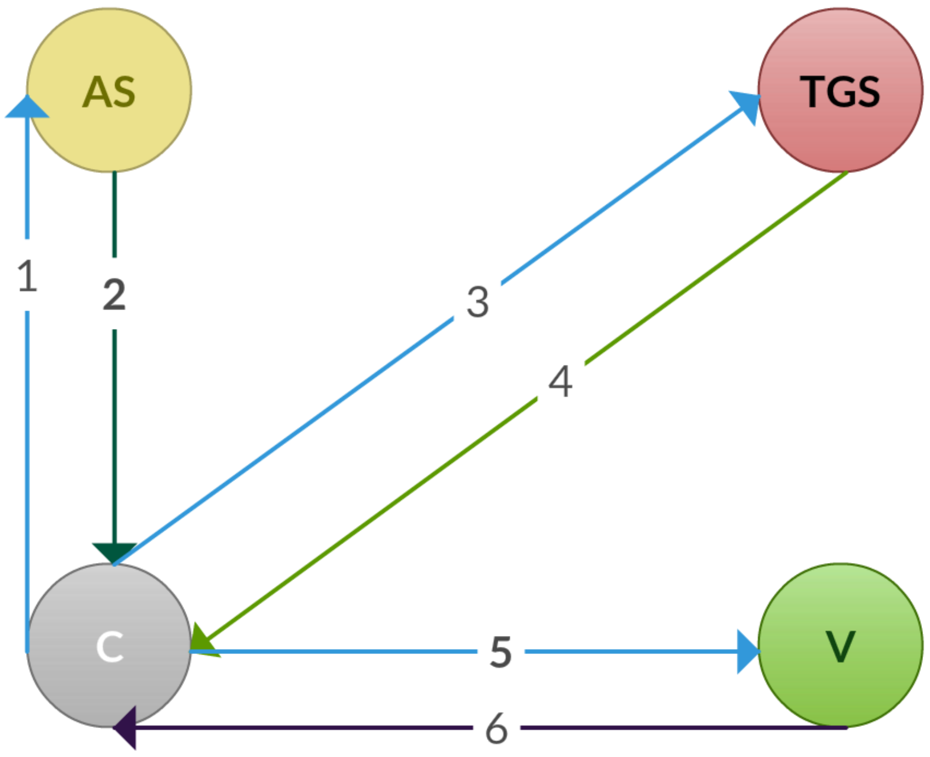


Figure 2.9 - Kerberos Protocol

The above figure shows the Kerberos protocol[23] in brief, which highlights the below steps.

AS - Authentication Server

TGS - Ticket Granting Server

TGT - Ticket Granting Ticket

SGT - Service Granting Ticket

V - Services Server

L - Lifetime of the ticket

T - timestamp at the moment

KDC consist of AS and TGS

1. $C \rightarrow AS$: C, TGS, n_c

Client sends the relevant user’s or the own ID C as per the use case(the ID of the authenticating entity), the ID of TGS and a nonce value.

2. $AS \rightarrow C$: $eK_{cs} (K_{c,tgs}, n_c, L_1, TGS), ticket_{C,TGS}$

AS lookup for the presence of the ID sent by the client and issue $K_{c,tgs}$ the session key to be used between C and TGS encrypted by the secret of the client known to AS, and $ticket_{C,TGS}$,

$ticket_{C,TGS} = eK_{tgs}$ ($K_{c,tgs}$, C, L_2) which carries the session key between C and TGS encrypted by the key shared by the AS and TGS.

3. $C \rightarrow TGS$: $ticket_{C,TGS}$, $eK_{c,tgs}$ (C, T_c), V, n_c'

The client sends the received $ticket_{C,TGS}$ as it is, to the TGS along with own ID and timestamp encrypted with received session key.

4. $TGS \rightarrow C$: $ticket_V$, $eK_{c,tgs}$ (n_c',L_2, V, $K_{c,v}$)

TGS replies with the session key to be used with communications with V, encrypted with session key between TGS and C, along with $ticket_V$ as below.

$ticket_V = eK_{VS}$ ($K_{c,v}$, C, L) where K_{VS} is the key between AS and V. This is referred to as the SGT.

5. $C \rightarrow V$: $ticket_V$, $eK_{c,v}$ (C, T_c')

The client initiates communication with V, sending the received $ticket_V$ and own ID and timestamp encrypted with the session key to be used with V, issued by TGS.

6. $V \rightarrow C$: $eK_{c,v}$ (T_c')

V should then decrypt SGT with its secret key, derive the $K_{c,v}$ from it and decrypt the second message from C to derive T_c'. Service server will then respond to client encrypting the same with $K_{c,v}$, indicating successful authentication of service.

As seen in the flow, the password of the authenticating entity is never sent in the network, which is a major advantage of the protocol.

2.7.4. The platform provided privileged API based authentication

This can also be considered as a challenge-response authentication mechanism, a special favor of it.

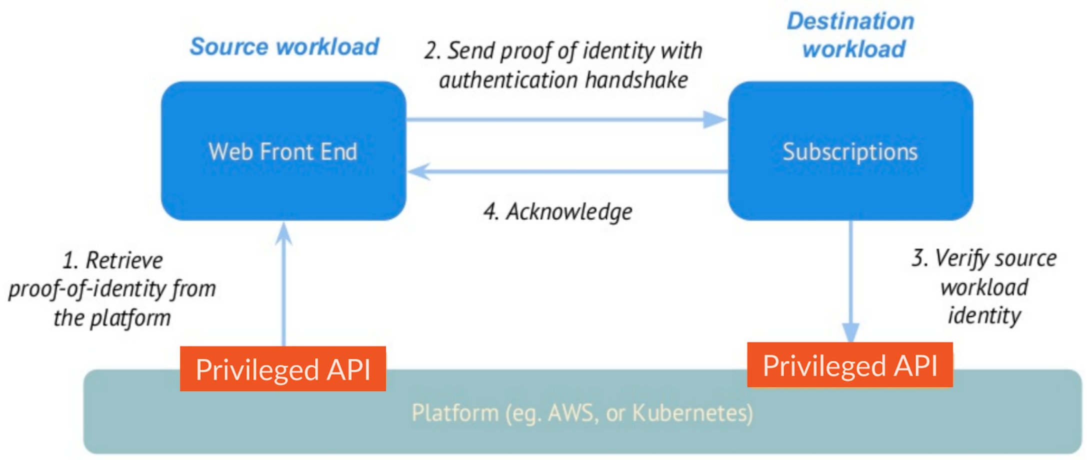

Source workload
2. Send proof of identity with authentication handshake
Destination workload
Web Front End
Subscriptions
4. Acknowledge
1. Retrieve proof-of-identity from the platform
3. Verify source workload identity
Privileged API
Privileged API
Platform (eg. AWS, or Kubernetes)

Figure 2.10 - Platform Mediated Authentication

The challenge is to provide a valid proof-of-identity document provided by the platform where both the source and destination workloads are running. As the source workload is running on the platform, it can validate a set of parameters unique to the workload to issue an identity document for that workload. The destination workload can ask the same platform to validate it. This is done through a special privileged API provided within the platform for the purpose. The limitation within this is that both the workloads should reside within the same platform as seen from the figure 2.10 from a presentation done by SPIFFE community or should understand how to validate the document provided by another platform.

Below we discuss 3 such privileged APIs provided by the industry giants Amazon, Google and Microsoft.

2.7.4.1. Amazon EC2 IID

Amazon Elastic Compute Cloud provides a privileged API to retrieve an Instance Identity Document(IID) within an instance. IID is a JSON document is signed so that its content integrity is preserved while authenticity, origin details provided on the claiming information. This can be validated using the Amazon CA certificate for each region. Below is a sample IID content[24].

```
{
    "devpayProductCodes" : null,
    "marketplaceProductCodes" : [ "1abc2defghijklm3nopqrs4tu" ],
    "availabilityZone" : "us-west-2b",
    "privateIp" : "10.158.112.84",
    "version" : "2017-09-30",
    "instanceId" : "i-1234567890abcdef0",
    "billingProducts" : null,
    "instanceType" : "t2.micro",
    "accountId" : "123456789012",
    "imageId" : "ami-5fb8c835",
    "pendingTime" : "2016-11-19T16:32:11Z",
    "architecture" : "x86_64",
    "kernelId" : null,
    "ramdiskId" : null,
    "region" : "us-west-2"
}
```

As in the above sample document, it consists of several attributes related to the instance based on its infrastructure, environment and service account belongs to. It is generated when the instance is launched and exposed through a metadata API accessible within the instance. Amazon recommends to retrieve this document frequently and validate due to the dynamic nature of the instances.

2.7.4.2. Google Cloud Provider IIT

Google Cloud Provider has a privileged API which issues identities for instances named Instance Identity Token[25]. Similar to Amazon, Google is also issuing a signed JSON with the instance details, but taking a step forward, they have made this an IDToken according to the OpenIDConnect 1.0 specification[26] which is as below, consisting of 3 parts ‘header.payload.signature’, separated by dots.

```
{
 “alg”: “RS256”,
 "kid": "511a3e85d2452aee960ed557e2666a8c5cedd8ae",
}.
```

```
{
  "iss": "[TOKEN_ISSUER]",
  "iat": [ISSUED_TIME],
  "exp": [EXPIRED_TIME],
  "aud": "[AUDIENCE]",
  "sub": "[SUBJECT]",
  "azp": "[AUTHORIZED_PARTY]",
  "google": {
   "compute_engine": {
    "project_id": "[PROJECT_ID]",
    "project_number": [PROJECT_NUMBER],
    "zone": "[ZONE]",
    "instance_id": [INSTANCE_ID],
    "instance_name": "[INSTANCE_NAME]"
    "instance_creation_timestamp": [CREATION_TIMESTAMP]
   }
  }
}
```

Google guarantees that an IIT can only be generated by the resident instance and no other instance can access it. Google cloud compute engine creates a unique token each time a request is made by an instance. Each of these tokens expire within an hour and is signed by Google’s public OAuth certificates for anyone to validate and trust.

2.7.4.3. Microsoft Azure MSI

Microsoft has renamed this previously known ‘Managed Service Identity’ to ‘Managed identities for Azure resources’ recently. They categorize identity management into two categories as,

- system-assigned managed identity - Azure creates an identity for the instance in the Azure AD. The identity of the instance is bound with the life cycle of the instance and gets deleted at instance deletion.
- user-assigned managed identity - This identity has an existence apart from the instance life cycle and can be assigned to multiple Azure service instances.

In the context of this research, system-assigned managed identity is considered.

Azure provides a broader view on the identity management of the instances as seen in the below diagram, providing options to govern different aspects of the flow.

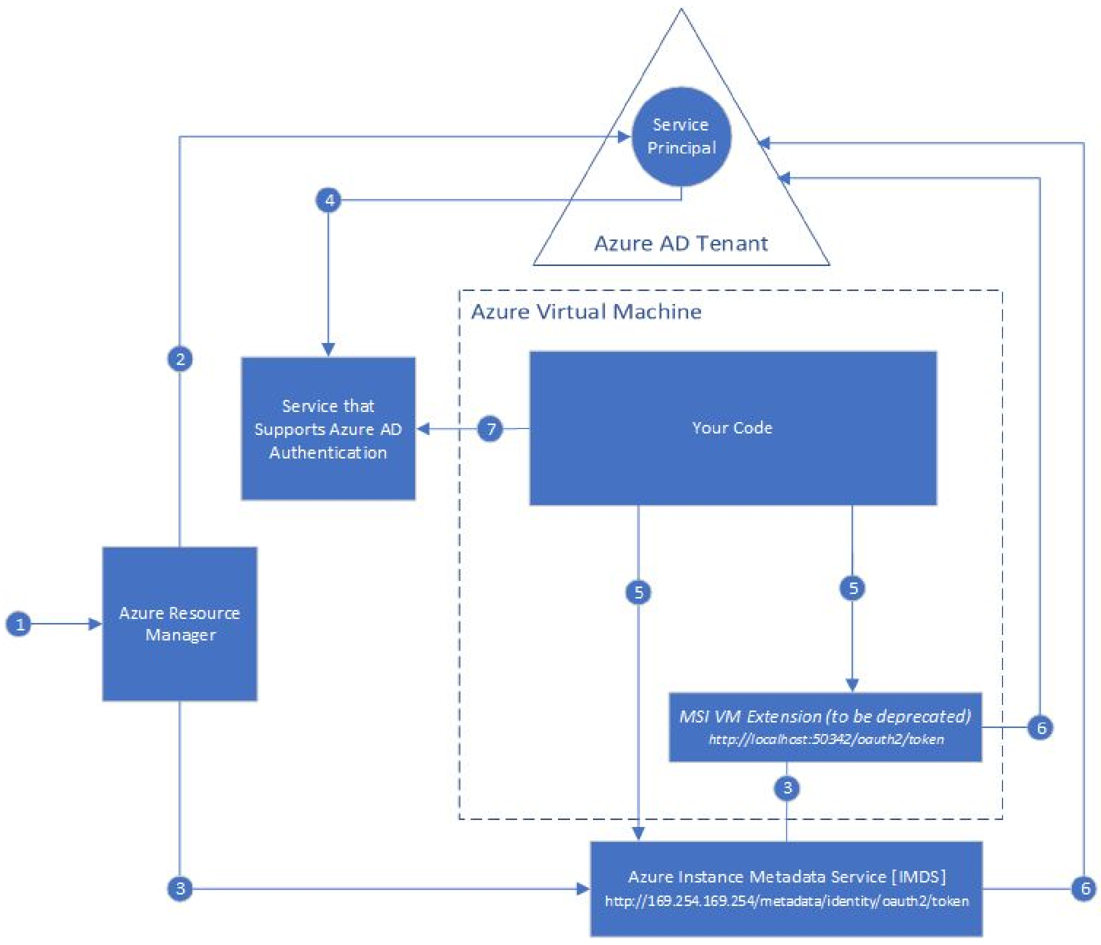

Service Principal
Azure AD Tenant
4
2
Azure Virtual Machine
Service that Supports Azure AD Authentication
7
Your Code
5
5
1
Azure Resource Manager
MSI VM Extension (to be deprecated)
http://localhost:50342/oauth2/token
6
3
3
Azure Instance Metadata Service [IMDS]
http://169.254.169.254/metadata/identity/oauth2/token
6

Figure 2.11 - Azure Instance Authentication

Similar to Google, Azure also make use of Open standards in the process, issuing the identity document in the format of a self-contained JSON Web Token(JWT), based on OAuth 2.0 standard[27]. Along with assigning identities, Azure also allows assigning roles to these instances for it's RBAC functionality, in addition to authentication.

2.7.5. SPIFFE standard

With the enterprises going digital and movements like Open Banking forcing domains to go digital, more and more enterprise system will get introduced into the world and it is inevitable that these systems will have to interoperate. With these numerous systems having to interact to deliver a service, Zero-trust architecture with roots of ‘never trust, always verify’ have gained more attention, to stress on the security of the systems. SPIFFE (Secure Production Identity Framework For Everyone)[28] is a standard initiated with the intentions of bootstrapping and maintaining trust related validations on a zero-trust system for workload authentication. This is a standard accepted by the CNCF and it steered by engineers from industry giants such as Google, Dropbox, Docker, and Scytale.

SPIFFE defines an architecture which removes the platform locking limitation of using the platform provided privileged authentication mechanisms and open doors to authentication between the platforms as well. In order to support this architecture, SPIFFE defines 4 standards as below.

- The SPIFFE Identity and Verifiable Identity Document(SVID) - Specifies an identity issuance standard, defining the required components in the process.
- The X.509 SPIFFE Verifiable Identity Document - This defines the details of a previous standard’s mentioned SVID format based on X509 certificate format.
- The JWT SPIFFE Verifiable Identity Document - This defines the details of the SVID in the JWT format.
- The SPIFFE Workload Endpoint - Defines the details of an endpoint that provides a set of gRPC methods which can be used by workloads to retrieve SVIDs and their related trust bundles.
- The SPIFFE Workload API - The exact set of services provided by the above endpoint is defined in this specification.

2.7.5.1. SPIFFE in action

SPIFFE standard defines a flow as shown below in figure 2.12 for trust bootstrapping. This is based on the reference implementation of SPIFFE, named

SPIRE (SPIFFE Runtime Environment). Below is a description of the involved components.

1. **Identity Registry** - SPIRE server has an own identity registry which keeps two coarse-grained attributes that decides how the SPIFFE IDs will be issued to a workload. It keeps details as in the below table.

SPIFFE ID	Node Selector	Process Selector
spiffe://abc.com/bill	aws:ec2:1234	k8s:namespace:1234
spiffe://xyz.com/account	token:7236427472	unix:uid:1002

Table 2.1 - Attestation Policy for SPIFFE IDs

A separate registration API is provided to manage these entries in the identity registry.

2. **Node Selector** - This defines a machine (physical or virtual) where a workload can be running on. The exact type of selector to be used is decided based on the infrastructure provider (AWS, GCP, bare metal) that the workload is running. Eg. AWS EC2 Instance ID, a serial number of a physical machine. Node attestor act based on the infrastructure provider to honor their selectors.

3. **Workload Selector** - This defines how to identify a process as representing a workload after the node is identified. This can be described in terms of attributes of the process itself (eg. Linux UID) or in terms of indirect attributes such as a kubernetes namespace. The node agent is responsible to verify that a particular process on a machine qualifies for its workload selector. Workload attestor act based on the process attributes to honor the process selectors.

4. **SPIRE Node Agent** - A process that sits on the node, verifies the provenance of workloads running on the node and provides those workloads with certificates via the Workload API, based on the selectors.

The trust bootstrapping process is illustrated below involving the above components.

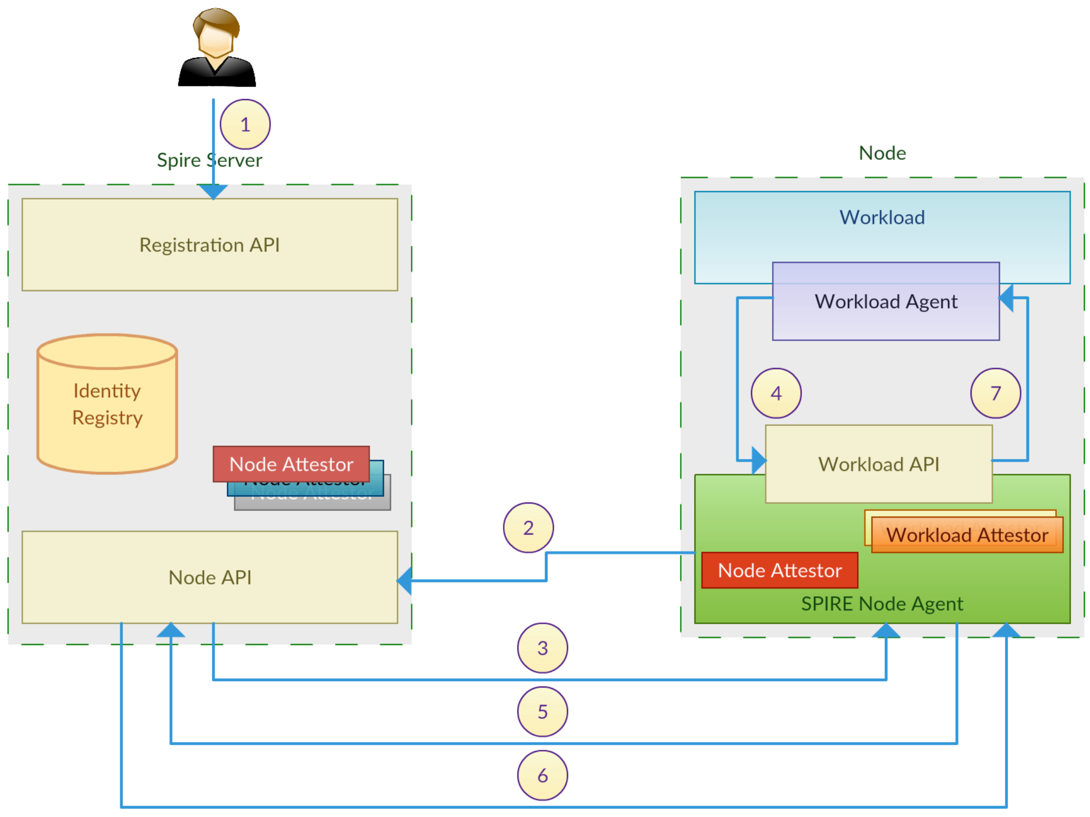


Figure 2.12 - SPIFFE in action

1. Registration API is called by either an administrator or a third party application to populate the identity registry with the required SPIFFE IDs and relevant selectors.
2. Node agent gets authenticated with the SPIRE server using a pre-established cryptographic key pair or based in the infrastructure provider. For example in the case of AWS EC2, node agent will submit the node's Instance Identification Document(IID) issued by AWS.
3. Node attestor in the SPIRE server validates the provided identification document based on the used mechanism. If the AWS IID is used, the relevant

attestor will validate it with AWS settings. Upon successful validation, the SPIRE server sends back a set of SPIFFE IDs that can be issued to the node along with their process selector policies.

4. When workload start to run in the node, it first makes a call to the node agent asking ‘who am I?’.
5. Based on the process selectors node agent received in the previous step, and using the workload attestors, the agent decides on the SPIFFE ID to be given to workload. It generates a key pair based on that and sends the CSR(Certificate Signing Request) to the SPIRE server.
6. SPIRE server responds to the node agent with the signed SVID for the workload along with the trust bundles, indicating which other loads can be trusted by this workload.
7. Upon receiving the response from SPIRE server, node agent, handover the received SVID, trust bundles the generated private key to the workload. This private key never leaves the node its workload belongs to.

2.7.5.2. SPIFFE implementations

- SPIRE - The reference runtime implementation provided by ‘Scytale Inc.’ which has initiated SPIFFE effort studying similar systems that are in production at Google, Netflix, and Twitter.
- Istio[29][30] - They have implemented part of SPIFFE standard to bootstrap identity, but have spread out focus on identity mgt, monitoring, throttling in a service mesh architecture specifically.
- Linkerd - Another service mesh platform provider similar to Istio, that is also looking into implement SPIFFE standard into their platform[31].
- Kubernetes Container Identity Working Group - They are also focusing on the SPIFFE standard, under discussion level. This WG has also shown interest in a vendor-agnostic way for workload authentication so that they can also interoperate[32].

From above SPIRE is much lighter weight having the sole focus on trust bootstrapping and workload authentication as the primary goals.

2.8. Workload Authorization Technologies

For Authorization, several options exist such as RBAC, ABAC, policy-based access control and OAuth 2.0 token based access delegation.

	RBAC	ABAC
Simplicity	Yes	Can be Complex
Fine-grained	No	Yes
Standardized	No	Yes (XACML/OPA)

Table 2.2 - Access Control Technologies Comparison

From these options the OAuth 2.0 framework is standardized and it the de-facto standard used by many of giants in the industry and enterprises for access delegation. It can also be combined with other access control options to provide fine-grained access control.

At the scale and dynamic nature, we are dealing with in the system, manual registration of clients as OAuth 2.0 applications is not very viable. There are two options present in this case.

- Use the OAuth2 DCR endpoint to register the workloads as clients before the tokens are requested.
- Create a client at the occasion of the token request, based on the SVID X509 certificate content used in establishing the mTLS connection.

Hence the authentication happened in the above approach will be consumed in this step itself again and an OAuth 2.0 access token needs to be issued. We will try to use the concept described in the draft standard on "OAuth 2.0 Mutual TLS Client Authentication and Certificate Bound Access Tokens" and apply the same in a highly dynamic and scaling environment.

This approach will let any of the services that have an MTLS connection with the authorization server to consume OAuth 2.0 based authorization.

2.8.1. OAuth 2.0 Authorization Framework

OAuth 2.0[27] is currently a widely used access delegation and authorization protocol and is the successor of OAuth 1.0[33] protocol. Flickr's authorization API and Google's AuthSub has inspired OAuth 1.0 standard which was released in 2007. Developers have been facing challenges with it due to its crypto-implementation and crypto-interoperability, which has led to the OAuth 2.0 framework backed by industry giants such as Yahoo, Facebook, Salesforce, Microsoft, Twitter, Deutsche Telekom, Mozilla and Google at 2012.
Below is a brief comparison of OAuth 1.0 vs OAuth 2.0.

As seen above OAuth 2.0 is not a backward compatible version of OAuth 1.0 and can be considered a complete rewrite of the specification.

2.8.1.1. OAuth 1.0 vs OAuth 2.0

1. OAuth 1.0 does not depend on the transport layer delegating transport level security to HTTPS/TLS. But handles it through digital signatures for integrity and authenticity validations. On OAuth 2.0, it simply delegates that to be handled by HTTPS/TLS protocol and focus on access delegation. This has made it more developer friendly though there are criticisms on lowered security. OAuth 2.0 protocol is also trying to increase it's security measure further with efforts on specifications such as 'OAuth 2.0 token binding'[34]

and ‘OAuth 2.0 Mutual TLS Client Authentication and Certificate Bound Access Tokens’[35].

2. As OAuth 1.0 is based on cryptography OAuth 2.0 is not dependent on cryptography they have different flows defined and validation mechanisms.

While OAuth 1.0 has defined 2-legged between the user and resource server and 3-legged authorization between, user, client and resource server, OAuth 2.0 defines 4 formal grant types and several extended grants.

2.8.1.2. OAuth 2.0

OAuth 2.0 Authorization Framework identifies 4 roles involved in the flows it defines, as below.

“**Resource owner** - An entity capable of granting access to a protected resource. When the resource owner is a person, it is referred to as an end-user.

Resource server - The server hosting the protected resources, capable of accepting and responding to protected resource requests using access tokens.

Client - An application making protected resource requests on behalf of the resource owner and with its authorization. The term "client" does not imply any particular implementation characteristics (e.g., whether the application executes on a server, a desktop, or other devices).

Authorization server - The server issuing access tokens to the client after successfully authenticating the resource owner and obtaining authorization. ”[27]

It also defines 4 grant types as below.

1. Authorization code grant
2. Implicit grant
3. Resource Owner Password Credentials Grant
4. Client Credentials Grant

As per OAuth 2.0 specification own abstract, “The OAuth 2.0 authorization framework enables a third-party application to obtain limited access to an HTTP service, either on behalf of a resource owner by orchestrating an approval interaction between the resource owner and the HTTP service, or by allowing the third-party application to obtain access on its own behalf.” ,

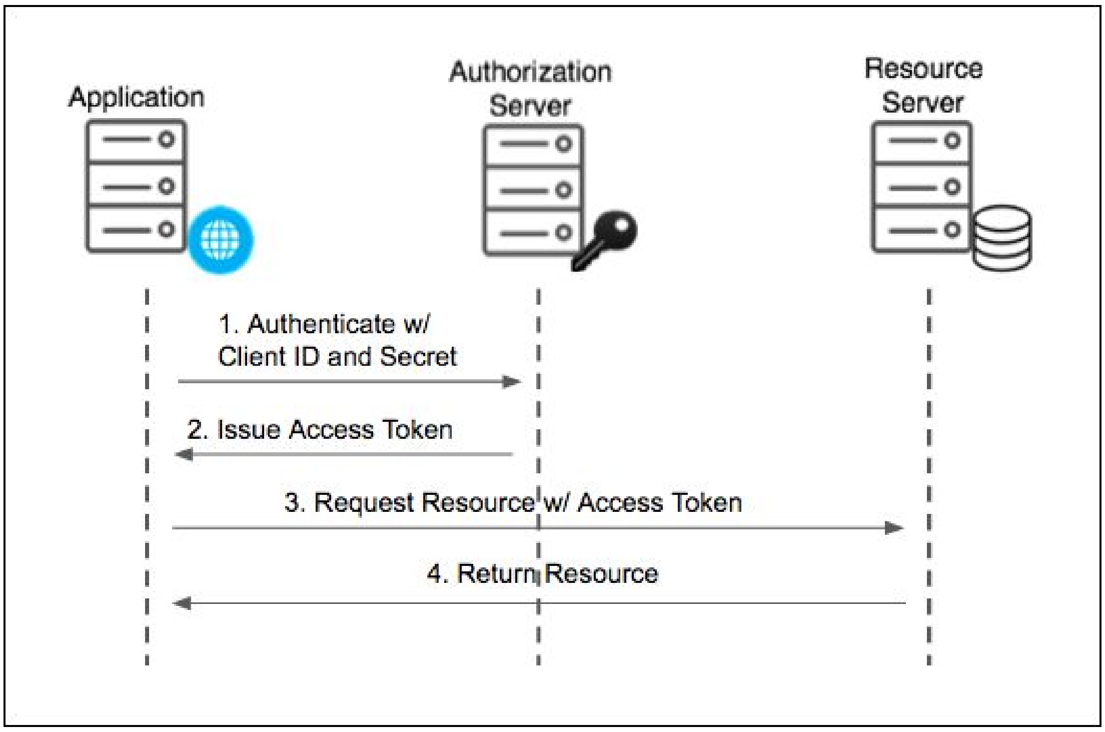


Figure 2.13 - Client Credentials Grant

The first 3 grant types orchestrate an interaction with resource owner while ‘Client Credentials’ grant obtain an access token for its own use, which can be applied for workload authorization. This grant type treats the client similar to a user and validates the credentials issued to the client by an authorization server.

As seen from the above figure 2.13, the authorization server issues an opaque string named ‘access token’ to the client application to be used to access resources from the resource server. Upon receiving a resource request along with an access token, resource server consults authorization server, on the validity of the access token and whether it has the scope to access the requested resource and act based on the feedback. This token referred to as ‘Bearer token’ as anyone bearing the token will be able to access the resource. Hence there have been debates on totally delegating security of the token to transport layer, because, if the attacker gets hold of the token, they fully leverage the access allowed on the token without any issue. As a remedy for this concern, some implementations make use of ‘self-contained access tokens’ which are signed and contain additional details on the bearer within the token string itself. Sometimes the token is made bound to the transport layer connection made by the client as in ‘OAuth 2.0 token binding’ draft.

2.8.1.3. Fine-grained authorization with OAuth2 scopes

As this research is focused on authorization between systems, which is close to client credentials grant type use case in OAuth 2.0, scope usage is investigated with relevant to that grant type in this section. The specification defines as scope as below and lets the authorization server decide on the scope selection implementation.

“The authorization and token endpoints allow the client to specify the scope of the access request using the "scope" request parameter. In turn, the authorization server uses the "scope" response parameter to inform the client of the scope of the access token issued. ”

In most of the industry implementations ‘scope’ attached to the access-token are decided based on RBAC concepts, while other approaches are not prevented.

Authorization Request

```
POST /token HTTP/1.1
    Host: server.example.com
    Authorization: Basic czZCaGRSa3F0MzpnWDFmQmF0M2JW
    Content-Type: application/x-www-form-urlencoded

    grant_type=client_credentials
    scope=clearance_level1 clearance_level0
```

Upon receiving the above request, authorization validates the provided client credentials in the ‘Authorization’ header. If it is valid, it further checks if this client can be granted the scopes it has requested, based on the available scope validating policy(eg: RBAC). After determining the allowable scope it generates the access token and stores it with the scopes the token was given, the validity period of the token and any other relevant attributes.

Authorization Response

```
HTTP/1.1 200 OK
    Content-Type: application/json;charset=UTF-8
    Cache-Control: no-store
    Pragma: no-cache

    {
      "access_token":"2YotnFZFEjr1zCsicMWpAA",
      "token_type":"example",
      "expires_in":3600,
      "scope":"clearance_level1"
    }
```

As in the above response, the authorization server has only the scope ‘clearance_level1’ for the token without allowing the ‘clearance_level0’ scope. Hence any calls the resource server receives with this access token ‘2YotnFZFEjr1zCsicMWpAA’ trying access resources under ‘clearance_level0’ will be rejected at the token validation.

2.8.1.4. OAuth 2.0 popularity

OAuth 2.0 framework has been very popular since its release in 2012, being the successor of OAuth 1.0. All most all the giants in the industry such as Google, Facebook, LinkedIn, Twitter etc, e-commerce sites such as eBay and Amazon.com make use of OAuth 2.0 as their API security protocol. The OpenID protocol, which is an authentication protocol based on OAuth 2.0 has also been adopted by these giants. The specifications that are getting published in the EU region to support recent initiatives such as OpenBanking are also based on OAuth 2.0 protocol including the FAPI(Financial API) similar to the HEART (Health Relationship

Trust) model also making use OAuth in the health care domain. Considering these facts, it is evident that the OAuth 2.0 has become the goto standard for API security in today's industry, while the future standards are also to be based on that.

2.8.2. Authorization Servers

1. KeyCloak - This is an open source IAM solution under Apache license, that supports all the well known IAM standards such as OAuth 2.0, OpenID Connect and SAML SSO. As per[36], they have plans to support MTLS based client authentication for OAuth 2.0, but currently supports certificate bound access tokens only.
2. Gluu - This is also an IAM solution under MIT license that supports IAM standards similar to KeyCloak. They are also considering MTLS supported OAuth 2.0 for future[37].
3. IBM API Connect - This is a closed source API Management solution that has inbuilt features to support OAuth 2.0 protocol in API access. They have recently added MTLS based OAuth client authentication support as in [38].
4. Ping Identity - They have a proprietary license based IAM solution for enterprises and supports OAuth 2.0 client registration based on MTLS certificate [39]. No materials found on MTLS based OAuth 2.0 token issuing.
5. WSO2 Identity Server - WSO2 provides an open sources solution that supports well-known identity standards similar to KeyCloak and Gluu. They have an initial version of MTLS based client authentication implemented[40]. As this is the initial version, it has only the basic functionality implemented with self-signed certificates used in MTLS.

3. SOLUTION DESIGN

This solution aggregates a number of different technologies and provides components that glue them to cater to the requirement of authorization for a dynamically scaling heterogeneous system, taking the ‘cloud’ as the most common example. While authorization is the main concern, authentication of the involved entities is also a crucial part of the flow that requires significant attention. The authentication approach of the solution was based on the literature review and the model is based on modern open security standards that have evolved from classical security models and architectural principles of cohesive components with separated duties and pluggable architectures.

The solution is named ‘Dvaara’ with the meaning of access control provided by doors and also has the meaning of opening doors of wider interoperability among heterogeneous systems.

When designing the solution, 4 possible approaches were considered to place authentication and authorization, as given below.

1. **Workload level authentication and authorization**

This is to authenticate and authorize requests coming to the workload, at the workload level itself (not necessarily by the workload, but maybe by a sidecar residing with the workload to manage authentication and authorization within the workload scope). While this allows the most fine-grained level of control on security, allowing the enforcement of verification close to the workload, in a dynamically scaling system, this will be much error-prone, because a new workload introduction, an update or a revoke will also need to be addressed at each workload level, adding the complexity of keeping them in sync. When the number of workloads under consideration is of a small size, this may be viable, but on the scale of a growing number of workloads, this will not be very practical.

2. **Workload level authentication and global authorization**

This proves to be fundamentally wrong, because, in order to perform authorization on an entity, its authenticity should have been verified first as authorization depends on the true identities. One approach to address this problem is to perform authentication at workload level and then go back to global scope and perform authorization, though it may not be very intuitive to expand the scope after authentication, that getting into a narrower scope. Still, this approach also has a problem that workload level authentication may not suffice to uniquely identify a workload in the global scope. For example at workload level, two workloads might identify themselves as 'workload-A' uniquely within their scope, but when those two entities reach the global level, it does not have a trusted way to identify these two separately.

3. **Global authentication and workload level authorization**

Global authentication is the best level of scope to verify authenticity because then the workloads get an identity that is valid globally and can be consumed at workload level authorization functionality as well. In this model workload still have to handle the authorization business logic, bearing the overhead of maintenance when the system scales. This deprives a single view on the authorization within the system, making it error-prone, as modification are made without being aware of the overall state of the system.

4. **Global authentication and global authorization**

In this approach, both authentication and authorization happen globally. Taking this responsibility to a global level keep the workload simple and lightweight while providing a central view and control on these aspects of the system. When the requirements raise, there may be cases where fine granular details of the workload will need to be considered at the global level, making it a bit more complex. Still, the level of complexity involved with that will be much less compared to managing that in workload level in larger scales.

With the above analysis, the solution is designed taking the path of global authentication and global authorization, while enabling the definition of fine-grained detailed based authorization policies at a global level in a central location.

In the below methodology section, the technology selection rationale and design decisions and assumptions are discussed in detail.

3.1. Methodology

When trying to address the problem of Authorization for a Heterogeneous system, the solution needs to be functional across these systems or at least provide clear extendability to make it functional across systems. With authorization is based on authentication, this requirement is valid for authentication mechanism of the solution as well. The below section addresses technology selection considerations of the solution.

3.1.1. Authentication technology

In the literature, there are 4 main authentication technologies. Below is an analysis of those that were considered in selecting the authentication technology of the solution for the workloads.

Mainly, 4 facts are considered as below. The most important requirement for the solution is that this authentication technology should work across platforms, as global authentication and authorization is the path selected for the solution. Then 3 more facts are considered based on the maintainability and strength of authentication. If the mechanism needs credentials to be stored with the workloads, then all the maintenance tasks come into the picture, such as secure key distribution and key rotation. Also if the workload can identify themselves with different identities at occasions to different parties that also add up complexity.

Mechanism	Do not require to deploy credentials with the workload	Single identity per workload	API driven credentials rotation and distribution	Cross-platform trust building
Firewall	Yes	Yes	No	Yes
Destination authentication	No	No	No	Yes
Platform mediated identity	Yes	Yes	Yes	No
SPIFFE	Yes	Yes	Yes	Yes

Table 3.1 - Workload Authentication Technology Comparison

While modern firewalls provide a lot of flexibility in defining authentication policies for the workloads, it doesn't provide seamless trust bootstrapping based on APIs, which is not much automation friendly. Also defining fine-grained authorization logic at the firewall level will overload the firewall unnecessarily and can affect the performance of all the traffic going through the firewall.

How a database client calls the database server to retrieve the data, can be considered a good example of the destination authentication. Here the database server generates a credential for the client and using a distribution mechanism get it delivered to the client. Then when the client calls the database the provided credentials need to be submitted and validated. This shared secrets mechanism work perfectly across platforms. Yet, the overhead of secure maintenance and distribution of credentials needs to be considered.

Platform mediated identity refers to identifications done by the OS, cloud platforms, middleware etc. For example, AWS has its own identity issuing mechanism with 'Instance Identity Document', which is valid in the AWS scopes. GCP has 'Instance

Identity Token' similar to this, which can be used to identify a workload in GCP world. Hence this solution lacks the most important requirement of the solution, which is to support cross-platform authentication.

As seen from the literature review, SPIFFE addresses all the considered functionality. It is built on top of platform mediated identity, hence has all the characteristics of it, added with cross-platform support. While the SPIFFE standard supports all the requirements, hence the SPIRE implementation as well, this is a fairly new standard. From the announcement of release at 2017 November at Kubecon, not much time has passed. Hence the community and background of the technology were also looked up. The standard is accepted for CNCF sandbox, which gives much credibility with the reputation of CNCF in the industry. The SPIFFE implementation is available in the GitHub under Apache 2.0 license, bearing a healthy number of contributors on their repositories. SPIFFE development community is also very active and helpful which was experienced their 'slack' channel. In addition, SPIFFE standard is used by "Istio" a Google Cloud solution based on Service Mesh Architecture, which adds more credibility. Analyzing the above functional and non-functional facts about SPIFFE, it was selected for the solution, despite being a very new protocol.

3.1.2. Authorization Technologies

3.1.2.1. DAC vs MAC

As discussed in the literature review, DAC and MAC approaches are available. While MAC has been used by the military grade systems, most enterprise systems use DAC approaches, where the authorization decision is taken at resource accessing and information flow afterward is not strictly addressed as in MAC. There are upcoming regulations such as GDPR in the European region that tries to enforce authorization on the information flow as well, based on user consent. Within the scope of this project, a DAC based authorization approach is suggested, while

keeping flexibility to integrate this type of policies that govern the information flow, in requirements such as GDPR compliance.
Below is a comparison of two access control approaches that can be used with DAC.

3.1.2.2. RBAC vs ABAC

When considering the workload authorization, there can be several attributes that need to be considered. These attributes may come from the environment they run on such as the host, user who initiated the workload, cloud platform provider etc. which are static. Additionally there can be dynamic attributes to be considered such as the uptime of the workload (maybe if policy needs to make sure enough time is elapsed to load all the configs from its start or a health check), current time (in case the policy want to skip a specific maintenance window of the workload) and other workloads it has communicated so far. This list of attributes can vary based on enterprise decisions and requirements, though it is evident that there are a lot of attributes to be considered in both static and dynamic manner. Hence the flexibility of ABAC is much required for the solution that been limited to RBAC.
Below is a comparison of two ABAC technologies currently used in the enterprise systems.

3.1.2.3. XACML vs OPA

Both of these protocols are very powerful and flexible in providing ABAC. Two sample policies are defined in Appendix 1 and 2, for the same policy, in these two protocols. As seen clearly from these examples, the XACML policy is based on XML and much verbose. The OPA policy is less verbose, looks much like JSON with few variations. Below is a brief comparison of the two standards.

	XACML	OPA
Flexible ABAC support	Yes	Yes
Extendability	Yes	Yes
Complexity	High	Occasionally
Verbose	Yes	No
Required training	Yes (Though it’s XML, have specific functions and behaviors to understand)	Yes (Though it’s JSON like, have special meanings for symbols and ways of writing rules
Implementation Availability	Axiomatics, Sun XACML engine, WSO2 Identity Server	OPA
Background	Open standard by OASIS	Open implementation under Apache 2.0 license, targeting cloud-native environments

Table 3.2 - ABAC Technology Comparison

Though XACML has been there for quite some time now, from its release in 2003, it hasn’t gained much popularity in the industry due to its complexity. There have been even discussions on whether ‘XACML is dead’[41]. It is still in use, being the de-facto standard for fine-grained authorization, with the least options available in the arena. Despite being a new technology released in 2017, OPA has come a long way now, being used by other cloud-related technologies such as ‘Kubernetes’ and ‘Istio’ and being accepted by CNCF. Considering the light-weightiness of the technology, the ability to satisfy the requirement and proven credibility despite been new, OPA was selected to be used in the project.

3.1.2.4. Authentication and Authorization

With the technology selections considered so far, SPIFFE is to be used for workload authentication and OPA is to be used to define the authorization policy. The next concern left to be addressed is the integration between these two protocols, defining how the SPIFFE identity can be consumed in an OPA policy and provide access control over a workload.

Approach 1

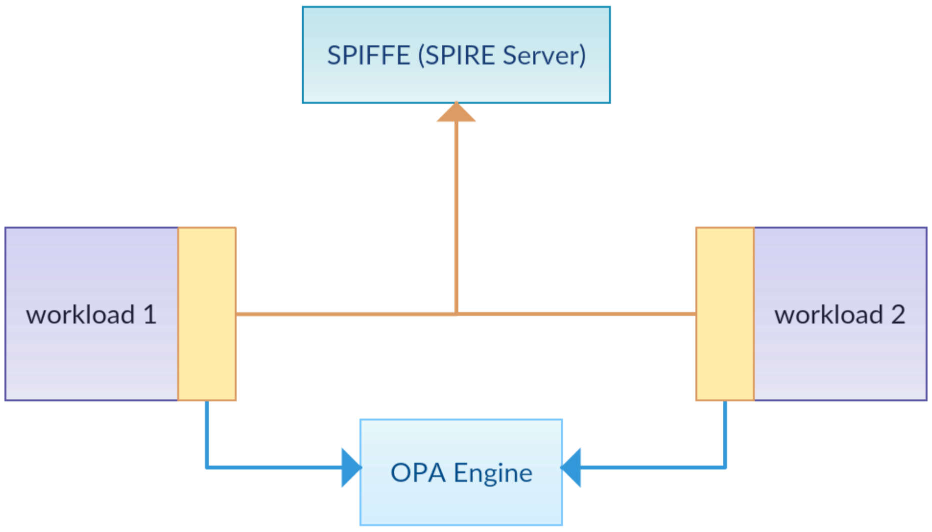


Figure 3.1 - Approach 1 for SPIFFE and OPA Integration

As in the above figure 3.1, let's consider workload 1 is running in cloud-A and workload 2 is running in cloud-B. If workload 1 needs to access a resource exposed by workload 2, then workload 1 should be able to get an identity issued by SPIRE Server and workload 2 should be able to validate that identity consulting the SPIRE server. Upon successful validation, workload 2 should check with the OPA engine to see the authorization level the workload 1 has over its resources and allow accordingly. The same process should be followed if workload 2 needs to access workload 1 as well.

Advantages :

- Workload has control over the authentication and authorization integration.

Disadvantages :

- Both the workloads should be able to communicate and understand SPIFFE protocol
- Mandatory to modify the workloads to do the above two calls, at least by injecting an interceptor in front of the workload.
- Need to develop plugins from scratch as suitable for multiple workload types
- There is maintenance overhead at each workload level to configure and maintain the above plugin

Additionally, the existing system designs need to be considered to see how the approach fit into the existing systems.

Below can be considered the most common design enterprise systems follow currently, with several authorization servers in the market, providing authentication, authorization, user mgt, identity mgt, SAML, OAuth kind of Identity and Access Management(IAM) capabilities.

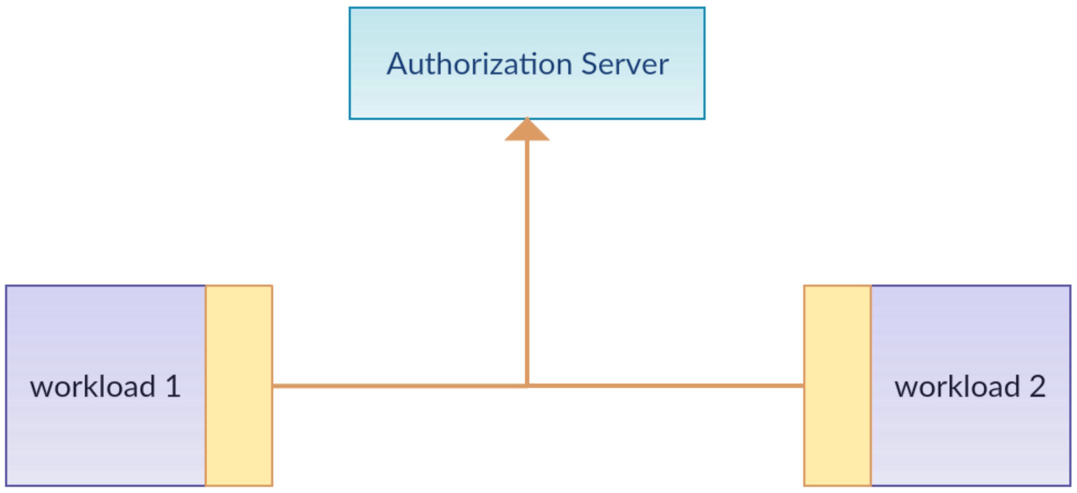


Figure 3.2 - Common Model

As discussed in the literature review, there are a lot of authorization servers that support this type of architecture, based on widely used open standards such as OAuth 2.0. This approach separates authentication and authorization management from the

workloads, while the authorization server centrally takes care of those functionality performing heavy processing. Since these open standards have been there for a while, there are already developed plugins that can be used at the workload level. For example, Node.js has an OAuth module named ‘passport-oauth2’, Angular.js has ‘angular-oauth2’ and Java-based implementations can use Apache Nimbus libraries. Also, there are widely available API-mgt solutions that be used to workload authorization, which add API security functionality with zero code.
Considering this already available design observed in the enterprise systems, below approach is suggested to minimize the changes required to the existing system and maximize adaptability of the approach.

<u>Approach 2</u>

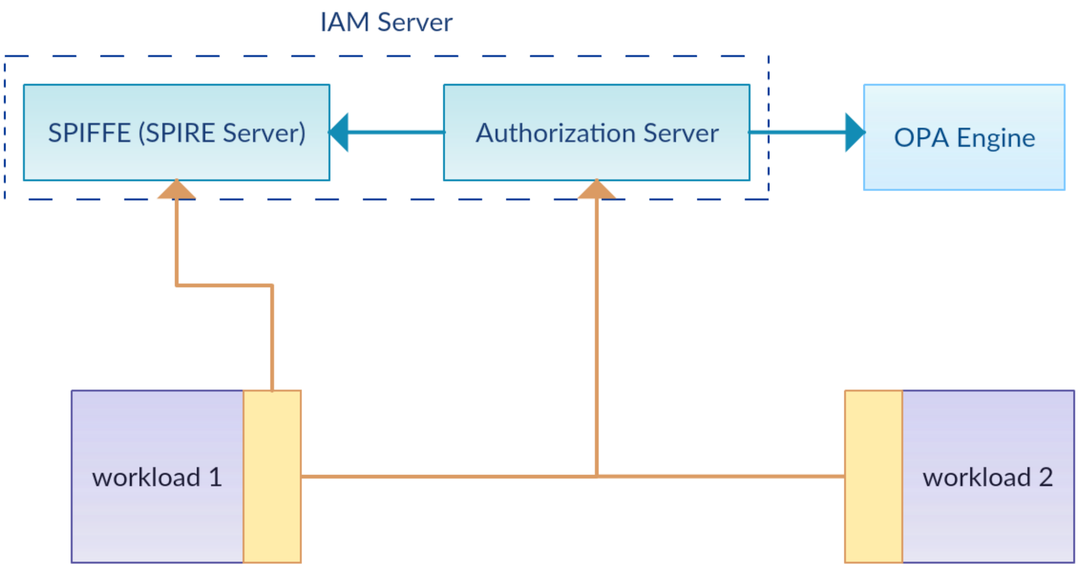


Figure 3.3 - Approach 2 with Authorization Server

In this approach, communication with the OPA engine is delegated to the authorization server and workloads can keep communicating with the Authorization server via the OAuth 2.0 protocol. The nodes running the workloads need to have a node SPIRE agent configured to communicate with the SPIRE server and obtain identity. These agents are currently available for current popular workload types such as workloads running on AWS cloud, GCP, a Linux machine or in a Kubernetes cluster.

Based on the characteristics of the workload 2 we can either join it to the SPIFFE world or keep it as a legacy system supporting OAuth 2.0 if it is not to scale dynamically. On the other hand, considering the IAM solutions in the middleware market today, embedding the SPIRE server into the IAM server should also possible to make an all-in-one solution. That is considered out of the scope of this project, moving forward keeping and running the SPIRE server separately.

3.1.2.5. Authorization Server

As discussed in the literature review there are several proprietaries and open source IAM solutions that can act as authorization servers. To support the above-discussed approach, it is mandatory that this authorization server provides comprehensive support for the OAuth 2.0 framework. Within the project scope, the solution is expected to be vendor neutral as much as possible, without depending on any vendor-specific features than extensions allowed in the OAuth 2.0 framework.

Based on the current familiarity of the author, the open source license and comprehensive OAuth 2.0 support, WSO2 Identity Server is used in the current solution. Yet the suggested approach is not bound to use only the WSO2 Identity Server.

3.2. Architecture

Below is the final suggesting architecture, based on the technology selection explained in the previous chapters. AWS and GCP are taken as examples for dynamically scaling, heterogeneous systems and there is a static system which does not scale dynamically. The call from workloads to the WSO2 Identity Server to obtain the OAuth2 access token is not shown in the figure to preserve clarity.

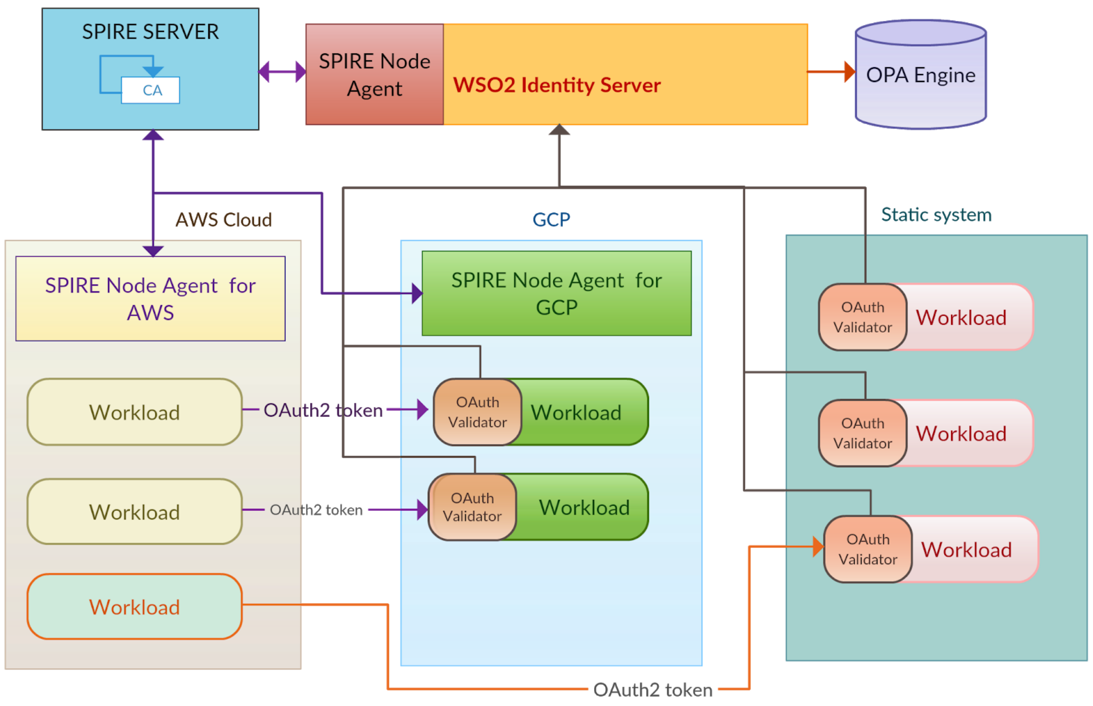


Figure 3.4 - Architectural Design

'Separation of duties' is followed in the architecture, giving each component a very specific task.

- SPIRE Server - Identify each workload in the system in a trusted manner
- WSO2 Identity Server - Exchange SPIFFE trust to OAuth 2.0 tokens
- OPA engine - Execute dynamic authorization policies

Each of these components is highly cohesive within the architecture and can be decoupled. If an innovative better option to replace any of the above components becomes available in the future, that component can be replaced with minimum effect to others. This is viable in the solution as the integrations are based on open standards, used by the majority of the industry. At the implementation stage also these properties are preserved which will be discussed in detail in the implementation section.

3.2.1.1. Interactions

The first step in the design is bootstrapping trust between dynamically scaling workloads of different types running on different clouds. The solution depends on the SPIFFE-based SPIRE server on this. At the end of this phase, all the systems in the SPIFFE world, including the WSO2 Identity Server which act as the authorization server, has got own unique identity across the clouds, similar to other workloads. This means each of these workloads has an identity and has received information on what other workloads to be trusted by the trust bundles sent from SPIRE server. Once the workload has the SVID (X509 certificate signed by the SPIRE Certificate Authority (CA), which is now trusted by the WSO2 Identity Server), it can exchange it to an OAuth 2.0 token under MTLS based OAuth client authentication specification. The scopes attached to this token represents the capabilities of the token, decided by the OPA engine. The workloads can consume these tokens to access a resource provided by another workload. Then that workload allows the access request, only if it is allowed as per the decision which was given by WSO2 Identity Server. In providing the decision, WSO2 Identity Server consults the OPA engine, providing the list of scopes attached to the token along with any other additional attributes available at the moment such as the accessing resource path using the token, token bearer details etc.

3.2.1.2. Assumptions

This architecture makes several assumptions on transport level security and workload attestation and node attestation mechanisms used by SPIFFE.

1. All the communications are done through secured communication channels. The OAuth2 tokens or any other information going through this transport in secured such that they can not be captured by unintended parties or even if they capture that doesn’t reveal any valuable information, that they can manipulate on.

    a. For this assumption to hold, the necessary precautions can be taken such as using stronger algorithms based keys with adequate key sizes etc.
    b. If token theft is calculated to be a greater risk, OAuth 2.0 token binding specification can be followed, which binds the token with transport level details. Hence even if anyone was able to steal a token, they won't be able to make use of it, as the authorization server will deny the request on token binding failure.
2. SPIFFE implementation relies on the underlying kernel or the platform to identify each workload uniquely within a node and the platform to identify each node uniquely. As the solution is based on SPIFFE, these assumptions hold for the solution as well.
    a. Using a combination of attestation policies and periodic or random checks on the attestation policy functionality can be precautions to be taken to make this assumption hold.

4. SOLUTION IMPLEMENTATION

The implementation of this solution 'Dvaara' is done, according to the design decisions made in the previous chapter. With the choice of WSO2 Identity Server acting as the authorization server in the design, its integration points are used to build up interoperability with SPIFFE standard and OPA engine.

The implementation needs to address two flows that run through the WSO2 Identity Server.

1. OAuth2 token issuing which validates the SVID to authenticate the client.
2. OAuth2 token validation which makes a decision to allow or deny the resource access request.

Chain of responsibility pattern could be observed throughout the design of the WSO2 Identity Server. The required implementations to work with SPIFFE and OPA engine are plugged into the WSO2 Identity Server as below, following the same the pattern. The overall flow with above two steps is shown in the below Figure 4.1.

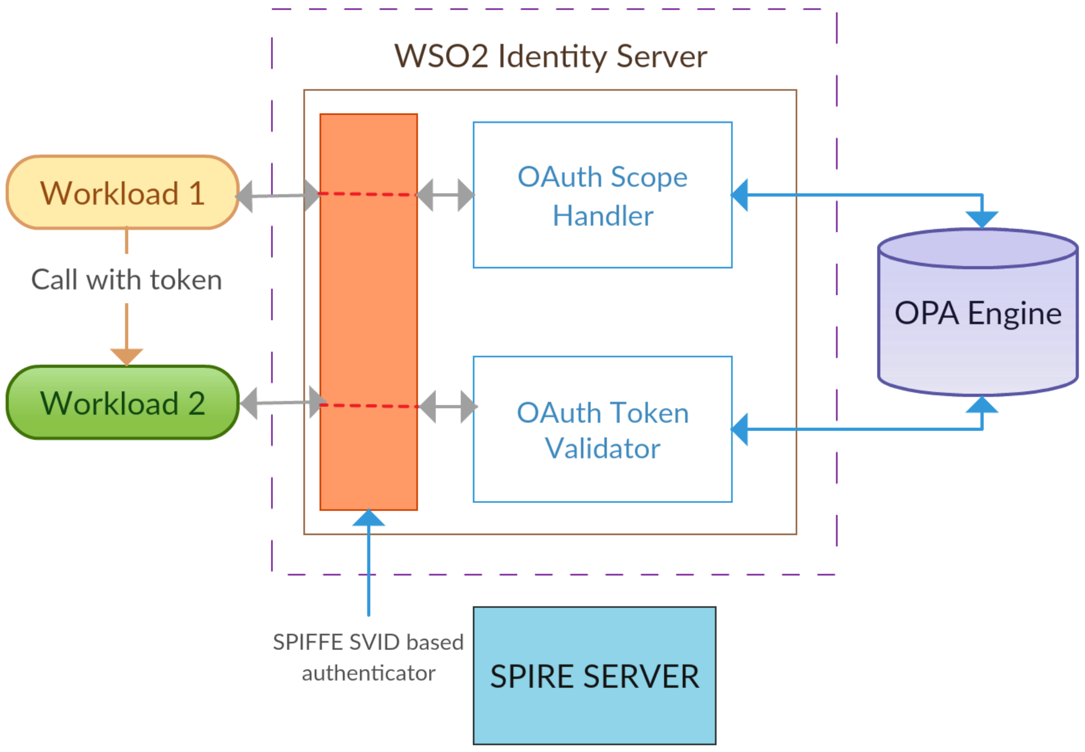


Figure 4.1 Implementation Scope

The area circled with broken lines defines the implementation scope of this project. Additionally, the WSO2 Identity Server was also improved to extract that relevant details from the transport layer and provide to the application layer as required, to be sent to the OPA engine as inputs for policy evaluation.

In brief, there is a separate SPIFFE security provider that is registered in the JVM to validate the certificates in the TLS layer that is used by an OAuth 2 specific authenticating artifacts developed for ‘Dvaara’. Further to that, a scope handler is present to decide on the scopes to be attached to the issuing token. Later when the access token is to be validated in use, a validator is engaged to handle OPA based validations whether to allow or deny the resource access.

4.1. Pre Resource Access - OAuth2 Token Issuing Flow

When a workload (workload 1) is to make use of another workload (workload 2), the first prerequisite is to have a valid OAuth2 token that is privileged to consume workload-2.

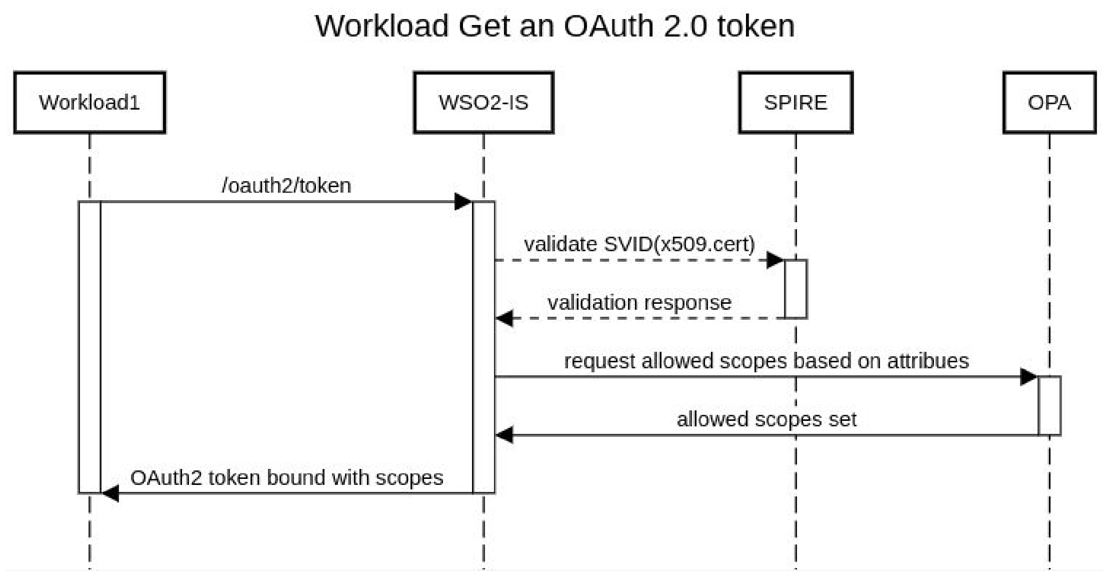

Workload Get an OAuth 2.0 token
Workload1
WSO2-IS
SPIRE
OPA
/oauth2/token
validate SVID(x509.cert)
validation response
request allowed scopes based on attribues
allowed scopes set
OAuth2 token bound with scopes

Figure 4.2 - Workload Gets an OAuth2 Token

To get a valid OAuth2 token this workload 1 calls the /oauth2/token endpoint making use of the SVID X509 certificate it received from SPIRE server, calling the SPIRE agent Workload API and going through the trust bootstrapping flow with workload attestation. By this time WSO2-IS which is also a part of the SPIRE trust network, it is received with the trust bundles, that instruct it on what to trust after the SPIFFE initial flow.

WSO2-IS validates the certificate used by the workload 1, in TLS connection creation. This is done making use of the 'java-spiffe' library[42] implemented by the SPIFFE community. This library extends the JAVA security API and provides SVID based KeyStore and TrustStore implementation. As WSO2-IS is based on tomcat server, this security provider is plugged in as a Tomcat HTTP connector, so that WSO2-IS will make use of SPIRE based trust and key management

Based on the OAuth MTLS specification[35] this certificate is used to validate that the requests come from a trusted client. If the validations on the certificates are passed, the implementation registers an application on behalf of the workload and let the request pass through the next chain of handlers in order to issue an OAuth2 token. To perform this validation and to act as a gatekeeper on the SPIFFE based validations, below detailed implementation at figure 4.3 is done, making use of WSO2-IS extensions.

The implementation is based on two interfaces from by WSO2-IS name ‘IdentityHander’ which is a generic handler in the chain of responsibility and ‘OAuthClientAuthentication’ which is a specific extension to validate requests coming from OAuth applications. As per the OAuth2 Framework specification, OAuth client authentication mechanism has a lot of flexibility. In this case, that flexibility is made use of while taking guidance from draft specification on MTLS based OAuth client authentication.

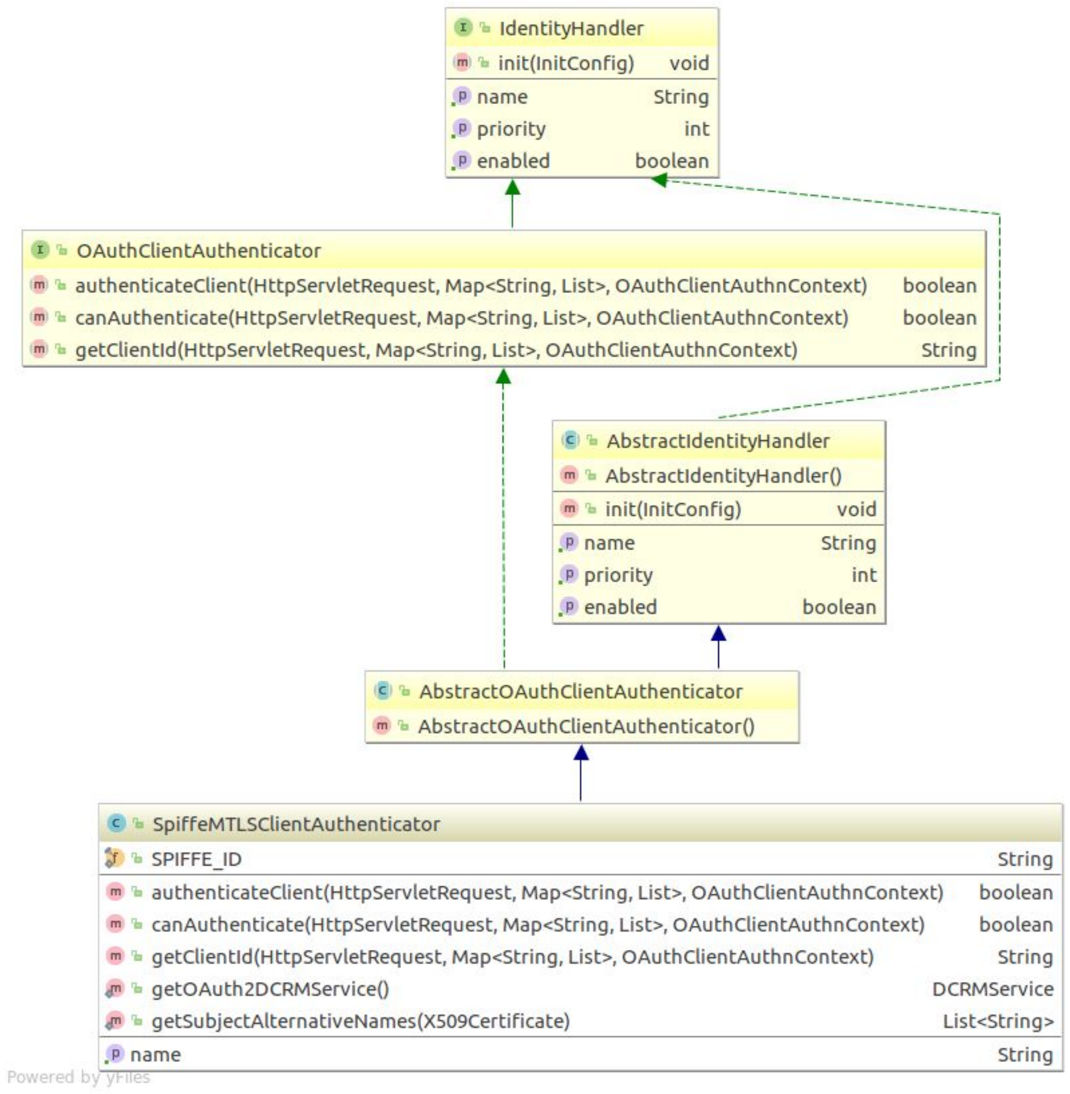

IdentityHandler
init(InitConfig) void
name String
priority int
enabled boolean
OAuthClientAuthenticator
authenticateClient(HttpServletRequest, Map<String, List>, OAuthClientAuthnContext) boolean
canAuthenticate(HttpServletRequest, Map<String, List>, OAuthClientAuthnContext) boolean
getClientId(HttpServletRequest, Map<String, List>, OAuthClientAuthnContext) String
AbstractIdentityHandler
AbstractIdentityHandler()
init(InitConfig) void
name String
priority int
enabled boolean
AbstractOAuthClientAuthenticator
AbstractOAuthClientAuthenticator()
SpiffeMTLSClientAuthenticator
SPIFFE_ID String
authenticateClient(HttpServletRequest, Map<String, List>, OAuthClientAuthnContext) boolean
canAuthenticate(HttpServletRequest, Map<String, List>, OAuthClientAuthnContext) boolean
getClientId(HttpServletRequest, Map<String, List>, OAuthClientAuthnContext) String
getOAuth2DCRMService() DCRMService
getSubjectAlternativeNames(X509Certificate) List<String>
name String
Powered by yFiles

Figure 4.3 - SPIFFE Based OAuth Client Authenticator

The implementation first checks if it can validate the request, verifying whether the required parameters are available. In this case, a check on the presence of a SPIFFE certificate (SVID) is done. Then at the authentication of the client, upon successful verification of SPIFFE certificate, it checks for the presence of an existing application or creates one if not and hands over the control to OAuth token issuing flow.

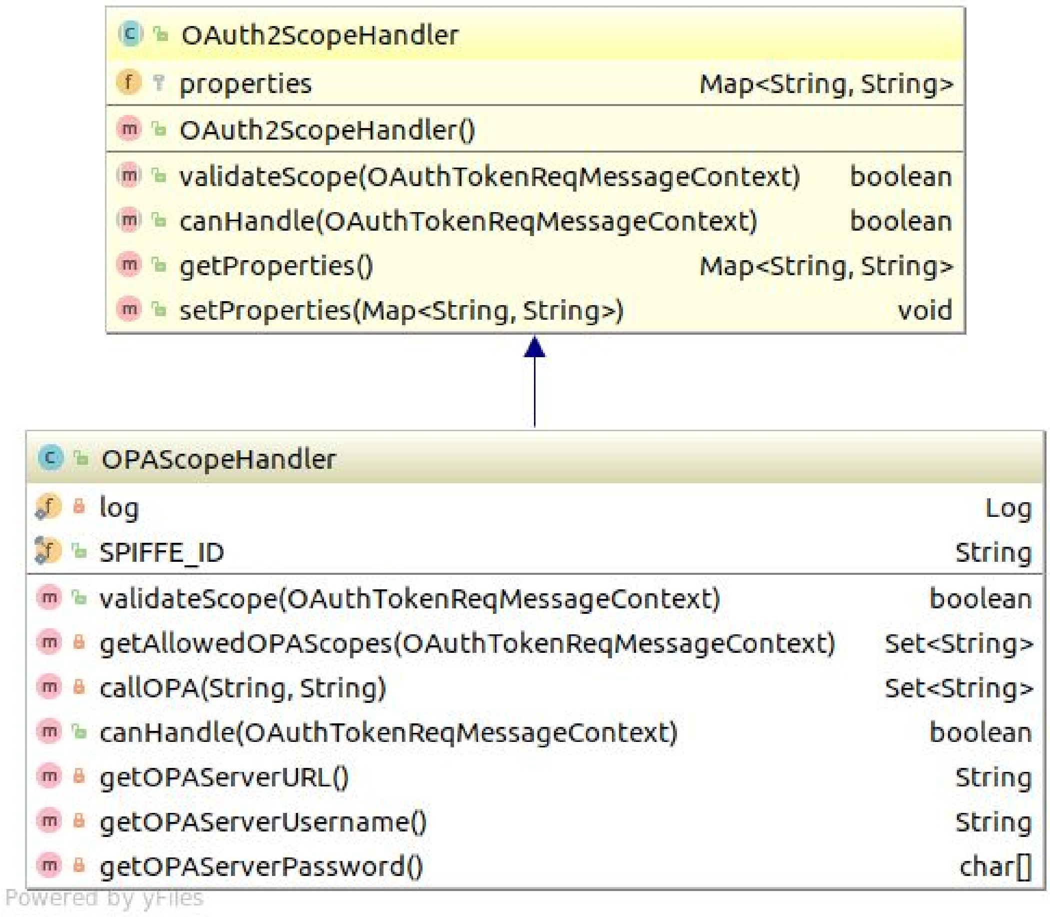


Figure 4.4 - OPA Based OAuth2 Scope Handler Implementation

This token issuing flow is intercepted by another handler as seen above, which decides on the OAuth2 scopes to be attached to the token, based on the decision from the OPA engine. In the very basic form, the SPIFFE ID is considered on deciding the scopes to be attached in the implementation. The OPA engine provides flexibility to dynamically define and modify this policy as per the business requirements.

The SPIFFE-ID represents an abstract level of verifications happened on the workload within the SPIRE network. When providing an identity to a workload, the SPIRE server and SPIRE agent perform several attestations as below.

While entry ID is a unique identifier used within SPIRE, other parameters represent the identification policy.

```
Entry ID:      bf6da55e-272d-4462-8d5d-bfa9567be752
SPIFFE ID:     spiffe://example.org/back-end
Parent ID:     spiffe://example.org/host1
TTL:           120
Selector:      unix:uid:1000

Entry ID:      32b27cd3-cb4c-43ce-a8af-d5dd344abc32
SPIFFE ID:     spiffe://example.org/front-end1
Parent ID:     spiffe://example.org/host2
TTL:           120
Selector:      unix:uid:1000

Entry ID:      a926e1cb-9989-4272-94a8-64528955c1b4
SPIFFE ID:     spiffe://example.org/front-end2
Parent ID:     spiffe://example.org/host2
TTL:           120
Selector:      unix:uid:1001
```

Based on the above attestation policy at SPIRE, when a workload is given the SPIFFE ID 'spiffe://example.org/back-end', it is confirmed that the workload is running in the host1 under Unix user id 1000 and will be getting SVIDs valid for 120s. When defining the OPA policy this information can also be considered as per the enterprise requirements.

For above verifications on the host and kernel-based characteristics, it relies on the platform based privileged API provided by the cloud providers that are discussed in literature review (AWS IID, Google IIT etc.) and kernels or container orchestration systems. The implementation trust those evaluations to be accurate.

SPIFFE ID	Node Selector	Process Selector
spiffe://abc.com/bill	aws:ec2:1234	k8s:namespace:1234
spiffe://xyz.com/account	gcp:7236427472	unix:uid:1002

Then for each SPIFFE ID, there are security labels attached. In the OAuth vocabulary, these will be scopes. These scopes provide an abstraction layer to consider in applying policies, mainly based on SPIFFE ID but can be extended to consider more attributes as below.

SPIFFE ID	condition	scope
spiffe://abc.com/bill	Health check passed and SVID expire> 3mins	clearance-0
spiffe://xyz.com/account	Health check passed and SVID expire > 1min	clearance-1

A given token can have multiple scopes with the definitions made. These scopes provide an abstraction to the defined verbose conditions.

In the simplest way below can be the representation of the policy in OPA engine.

```
{
 "scopes": [
   {
    "id": "spiffe:\/\/example.org\/wso2-is",
    "scopes": [
      "clearance-0"
    ]
   },
   {
    "id": "spiffe:\/\/example.org\/workload1",
    "scopes": [
      "clearance-2"
    ]
   },
   {
    "id": "spiffe:\/\/example.org\/workload2",
    "scopes": [
      "clearance-1",
      "clearance-3"
    ]
   }
 ]
}
```

This means if the SPIFFE ID of a particular workload matches any of these IDs, the OAuth2 token issued to those workloads can have the given scopes. Here the considered convention is clearance-0 is the highest privilege level, given to workloads with SPIFFE ID, ‘spiffe://example.org/wso2-is’, that will be eligible to consume any resources that need the highest privilege.

4.2. Resource Access - OAuth2 Token Validation Flow

Once the workload 1 gets a token as explained above, it will use this token to call the workload 2. Then workload 2 consults WSO2 IS, sending it the token and requesting whether to provide the requested access or not. WSO2-IS provides the response, checking the validity of the token and additionally checking on privileges based the scopes of the token and other attributes according to the defined OPA policy. This flow happens as shown in the below sequence.

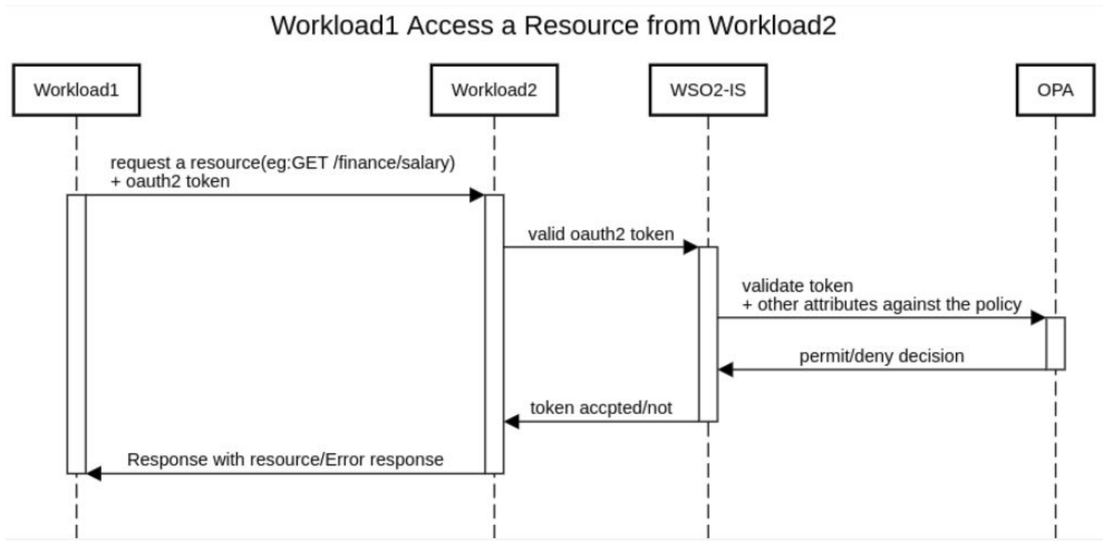

Workload1 Access a Resource from Workload2
Workload1
Workload2
WSO2-IS
OPA
request a resource(eg:GET /finance/salary) + oauth2 token
valid oauth2 token
validate token + other attributes against the policy
permit/deny decision
token accpted/not
Response with resource/Error response

Figure 4.5 - Workload Access Another Workload

The responsibility of taking this decision based on the decision of the OPA engine is given to another separate handler in the implementation. In addition to token and scope validations, it can perform fine-grained validations such as whether to allow the workload 1 to modify or read only a particular resource, exposed by workload 2.

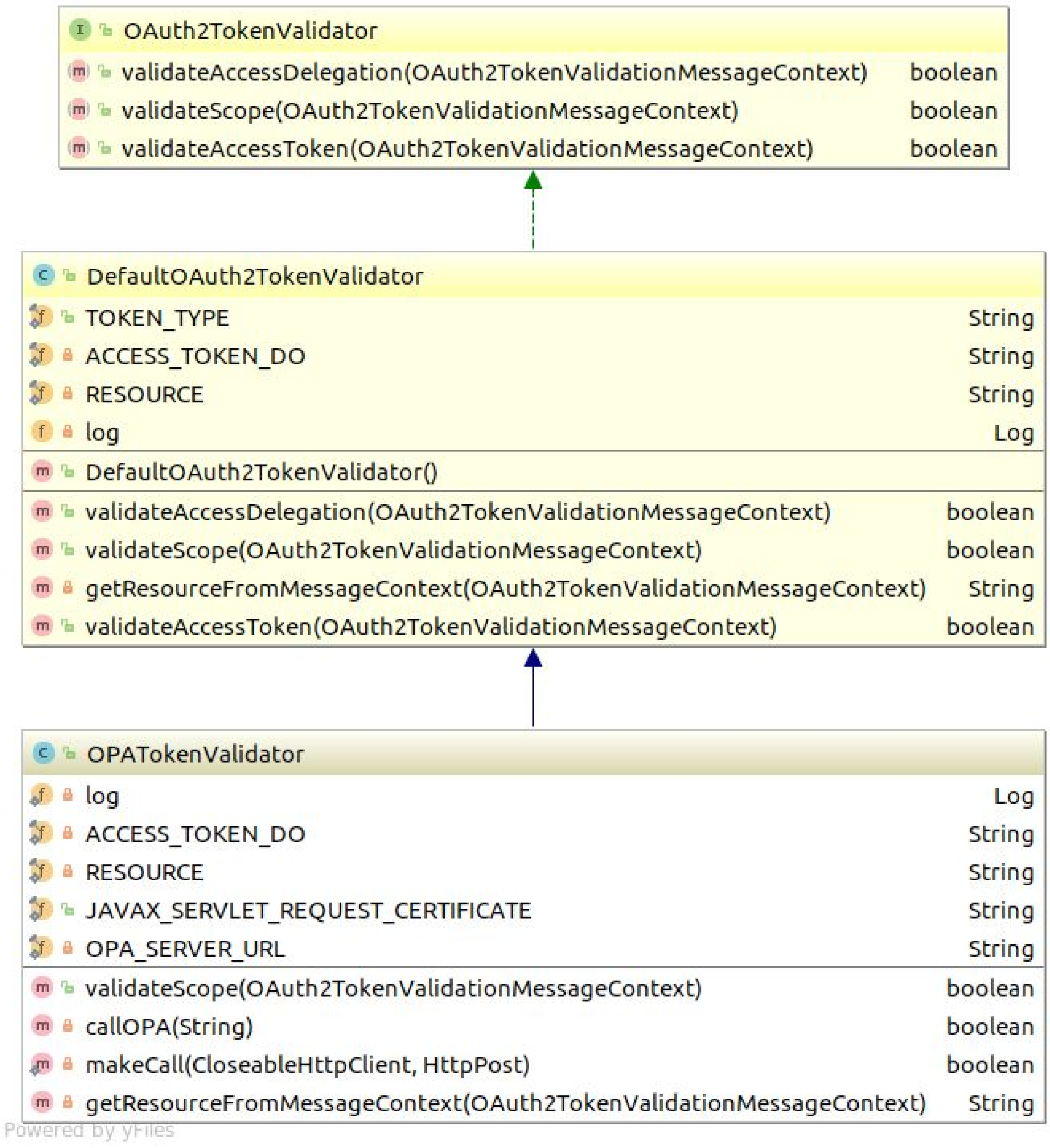

OAuth2TokenValidator
validateAccessDelegation(OAuth2TokenValidationMessageContext) boolean
validateScope(OAuth2TokenValidationMessageContext) boolean
validateAccessToken(OAuth2TokenValidationMessageContext) boolean
DefaultOAuth2TokenValidator
TOKEN_TYPE String
ACCESS_TOKEN_DO String
RESOURCE String
log Log
DefaultOAuth2TokenValidator()
validateAccessDelegation(OAuth2TokenValidationMessageContext) boolean
validateScope(OAuth2TokenValidationMessageContext) boolean
getResourceFromMessageContext(OAuth2TokenValidationMessageContext) String
validateAccessToken(OAuth2TokenValidationMessageContext) boolean
OPATokenValidator
log Log
ACCESS_TOKEN_DO String
RESOURCE String
JAVAX_SERVLET_REQUEST_CERTIFICATE String
OPA_SERVER_URL String
validateScope(OAuth2TokenValidationMessageContext) boolean
callOPA(String) boolean
makeCall(CloseableHttpClient, HttpPost) boolean
getResourceFromMessageContext(OAuth2TokenValidationMessageContext) String
Powered by yFiles

Figure 4.6 - OPA Based OAuth2 Token Validator

This implementation retrieves the scopes attached to the token and checks it against the requirements to access a particular resource in a workload. Though a common use related to REST APIs was considered as a workload in this implementation, this can be extended to any other type of workload, by writing a relevant OPA policy.

Assume a business policy as below on workload 2,

GET /finance/salary - clearance-3

POST /finance/salary - clearance-0

With policy enforcement done right at accessing the resources in the workload, it can more details to the OPA engine to decide on the request, such as the type of action to be performed, on which resource, user-agent used to make the request and any application layer details related the payload etc. This provides a lot of flexibility and control over authorizing the request.

Additionally, a client that can consume SPIFFE based trust and provide own SPIFFE based identity is implemented to evaluate the end-to-end flow under the given design. All the artifacts and implementation relevant with this solution resides at[43], been fully open to the community to use, provide feedback and contribute.

5. SOLUTION EVALUATION

5.1. Deployment Model

Dvaara approach was evaluated in a setup as shown below, which is a representation of a hybrid cloud. The configurations are maintained using 'docker' technology and published at the Dvaara GitHub repository[44].

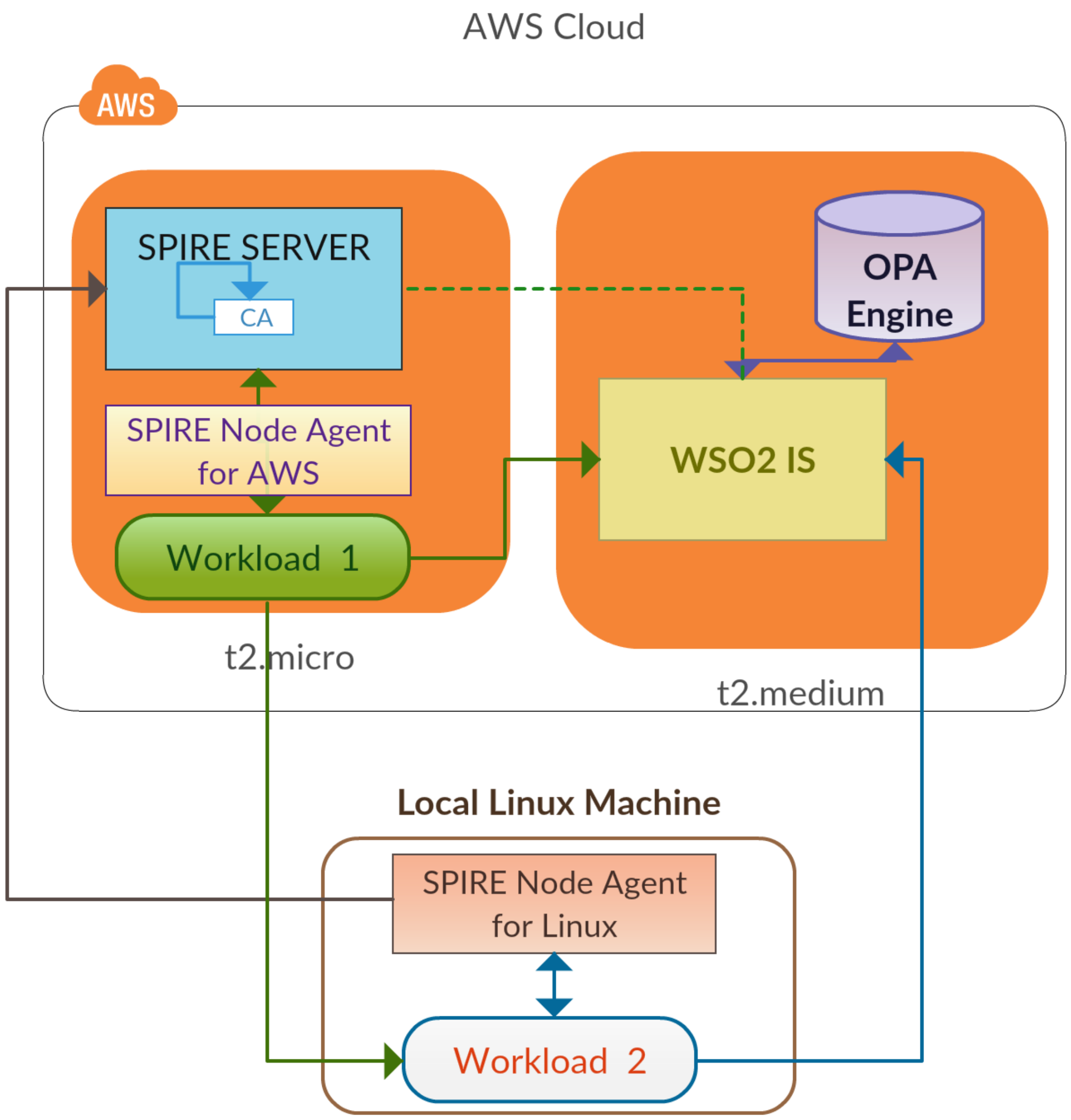


Figure 5.1 - Deployment for Evaluation

Following is the model used to provide proof of concept on the 'Dvaara' solution.

The test workload is a representation of an enterprise salary management system as a hybrid system.. This is modeled as Workload 2 in figure 5.1, which is designed as a REST service with the below definition. Company policy says:

- a user can read own salary details,
- their superior can modify the salary details.
- the user can't modify own salary details.

Reading and modifying salary details is done by the below REST calls respectively:
GET - /finance/salary/{username}
POST - /finance/salary/{username}

Assuming the company is transforming all its operations to a digital system, where they are to provide a single dashboard for its employees that is hosted in the cloud. As this requires integrating multiple software already running on heterogeneous systems, Dvaara solution is applied here. Workload 1 in figure 5.1 represents this dashboard which would be accessing the resources exposed by the Workload 2.

In the process, the legacy workload 2 is enriched with an access control layer in front of it, which extracts an OAuth 2.0 access token submitted by the consuming workload, sends it to the authorization server for validations and honors its decision to permit or deny access. The workload 1 should get a valid OAuth 2.0 token from WSO2 IS in order to submit in this call to workload 2. To satisfy the requirements to get an OAuth 2.0 token, workload 1 goes through the SPIFFE protocol and get a valid identity. In this model, the SPIRE server attests the node based on AWS IID and Unix user id of the process.

5.2. Deployment Configuration

5.2.1. Infrastructure

As seen from figure 5.1, SPIRE server and workload 1 are hosted in an AWS instance of model t2.micro. It accompanies a SPIRE node agent for that node. WSO2-IS and OPA engine are hosted in an AWS instance of model t2.medium as the WSO2-IS requires a minimum of 2GB memory as per the recommendation. The local machine runs the workload 2.

T2.micro

- 1 vCPUs
- 1 GB RAM
- OS: Canonical, Ubuntu, 16.04 LTS, amd64 xenial image build on 2018-11-14

T2.medium

- 2 vCPUs
- 4 GB RAM
- OS: Canonical, Ubuntu, 16.04 LTS, amd64 xenial image build on 2018-11-14

(T2.micro and T2.medium runs within the same AWS security group)

Local machine

- Intel® Core™ i7-7600U CPU @ 2.80GHz × 4
- 15.3 GB RAM
- OS: Ubuntu, 16.04 LTS (Xenial Xerus)

5.2.2. Policies

Two policies were defined in the OPA engine to make decisions as follows.

1. Based on the SPIFFE ID given to a workload decide on the clearance level they can be given. (This can be replaced in the future when SPIRE facilitates automating this.)

```
{
 "scopes": [
   {
    "id": "spiffe://example.org/wso2-is",
    "scopes": [
      "clearance0"
    ]
   },
   {
    "id": "spiffe://example.org/workload1",
    "scopes": [
      "clearance2"
    ]
   },
   {
    "id": "spiffe://example.org/front-end2",
    "scopes": [
      "clearance1",
      "clearance3"
    ]
   }
 ]
}
```

2. Based on the available environment parameters decide on whether to permit or deny a request, in addition to OAuth 2.0 token expiry validation done by the authorization server.

```
package httpapi.authz

subordinates = {"alice": [], "charlie": [], "bob": ["alice"], "betty": ["charlie"]}

# HTTP API request
import input as http_api
# http_api = {
#   "spiffe-id": "spiffe://example.org/front-end2",
#   "path": ["finance", "salary", "alice"],
#   "user": "alice",
#   "method": "GET"
#   "user_agent": "cURL/1.0"
#   "remote_addr": "127.0.0.1"
#   "iat":2019-02-10 12:27:27.196
# }

default allow = false
```

```
# Allow users to get their own salaries.

deny {
  http_api.method = "DELETE"
}

allow {
  http_api.method = "GET"
  http_api.path = ["finance", "salary", username]
  username = http_api.user
}

# Allow managers to get their subordinates' salaries.
allow {
  http_api.method = "GET"
  http_api.path = ["finance", "salary", username]
  http_api.scope = "clearance2"
  subordinates[http_api.user][_] = username
}

# Allow managers to edit their subordinates' salaries only if the request came from
a workload with SPIFFE ID "spiffe://example.org/workload-1"
allow {
  http_api.method = "POST"
  subordinates[http_api.user][_] = username
  http_api.path = ["finance", "salary", username]
  http_api.spiffe-id = "spiffe://example.org/workload-1"
}
```

This policy defines 4 rules.

- Deny any requests to delete
- Allow any requests coming to read the resource ‘finance/salary/{username}’ when it’s authenticated with the same user.
- Allow any requests coming to read the resource ‘finance/salary/{username}’ when the authenticated user is superior and used token has the scope ‘clearance2’.

- Allow modifying the resource ‘finance/salary/{username}’ only if the authenticated user is superior of the {username} and the request came from a workload with SPIFFE ID "spiffe://example.org/workload-1".

5.2.3. Test cases

5.2.3.1. Correctness

The correctness of the configuration is evaluated with the below cases.

Test case	Allow (yes/no)
Make a request to get an OAuth 2.0 token before joining the SPIRE network, from workload 2. (Should fail as not trusted)	No
Make the same above call after SPIRE trust is established.	Yes
Request a scope for the OAuth 2 token that is not allowed by the policy (spiffe://example.org/workload1 requests clearance 0)	A token is given without the scope
Request a scope for the OAuth 2 token that is allowed by the policy (spiffe://example.org/workload1 requests clearance 2)	A token is given with the scope
Alice read own salary calling GET on /finance/salary/alice	Yes
Alice delete own salary calling DELETE on /finance/salary/alice	No
Bob read Alice's salary calling GET on /finance/salary/alice with the previously received token (Allowed as Bob is Alice's superior and the token has scope clearance 2)	Yes
Bob modify Alice's salary calling POST on /finance/salary/alice calling from workload 1 (Allowed as Bob is Alice's superior)	Yes
Alice read own salary calling GET on /finance/salary/alice with an expired token	No
Start another workload in the t2.micro node with the same attributes as workload 1 (scaling workload 1). Make the requests from this workload. (Should allow as that is attested identical to the previous workload and node agent should handle the trusted introduction in the network under SPIFFE protocol)	Yes

Table 5.1 - Test Cases

5.2.3.2. Performance

The end to end flow involved in the system has two main flows after the SPIFFE based trust bootstrapping and identification is established.

- OAuth 2.0 token retrieval (SPIFFE based client authentication and OPA scope handling policy is evaluated in this flow.)
- Resource access call (OAuth 2.0 access token validation and OPA based attribute evaluation to deny or allow the request to happen in this flow.)

In a practical situation, a workload will run the first flow, get an access token and keep using it until it's expiration happens. Based on this, response times were measured individually for each of the above flows.

Three shell scripts were run simultaneously while each one making 1000 requests in a synchronous manner to get the average response times.

Token Retrieval

Fastest response time = 0.5455s

Slowest response time = 1.104s

Average response time = 0.6636s

Token Validation

Fastest response time = 0.5445s

Slowest response time = 1.0915s

Average response time = 0.6293s

Based on the above values it was observed that while sometimes the requests have taken 2x time from it's fastest response time, most of the times the requests have been served in around ~0.6s. Also, the token generation is slightly slower than the token validation. In the implementation, the first flow to retrieve a token has two database calls to retrieve client registration details and then to store the issued access token. The second call to validate the token, only perform one database call to check validity. While there can be other affecting reasons this seems to have a contribution to the slight difference.

While the main objective of the project is to suggest a viable architecture to address authorization in a dynamically scaling heterogeneous system, the response times can also play a major role when the solution is to be applied on an environment based on microservices architecture, where time scale is on milliseconds. There is space within the solution to improve response times making use of caching mechanisms which will be considered for the future. In addition to that the communication link between the on-premise and cloud components were through the public network in the evaluation setup. Hence providing a dedicated cloud VPN with enough bandwidth can also be expected to reduce the response times.

6. CONCLUSION AND FUTURE WORK

6.1. Conclusion

This project addresses a requirement that has been raised with the rising scale of developments happening with the digital transformation in the industries. It was evident that enterprises are looking to break the boundaries for the maximum benefit, without being vendor locked. Multi-cloud enterprise systems are a result of this. For example, enterprises may run computing intensive tasks on one cloud, keep sensitive information processing in an on-premise cloud and use a SaaS-based system consuming an already established well-reputed vendor in the field. This raises new challenges in information security aspects starting from workload authentication, as the involved identities join the system from various origins.

This solution 'Dvaara' provides an approach to address workload authorization challenge across a multi-cloud enterprise system, based on the well-established protocols such as TLS and OAuth 2.0 and consuming cutting edge technologies such as SPIFFE and OPA. The research community has already provided a solution for workload authentication in a multi-cloud system with the introduction of SPIFFE standard. Dvaara takes a step forward based on this protocol to provide workload authorization. It builds a bridge between the SPIFFE based workload identification and OAuth 2.0 protocol, by providing an implementation that consumes SPIFFE based identity for OAuth 2.0 client authentication. This can also act as a mediation bridge between existing enterprise systems that are based on TLS and OAuth 2.0 to work with modern systems based on SPIFFE. In addition, the solution injects more fine-grained authorization capabilities to the system via OAuth 2.0 protocol 'scopes' concept making use of OPA engine. Without being narrowly focused on just an OAuth 2.0 based solution, this provides a flexible authorization architecture that can deal with fine granular details in making service authorization decisions.

Dvaara demonstrates a proof of concept on how an existing authorization server that supports OAuth 2.0 control can be enabled to provide workload authorization in a multi-cloud system. It achieves this by implementing 3 main components. An authenticator that bridges SPIFFE based authentication for OAuth 2.0, a scope handler that selects the OAuth 2.0 scopes based on an OPA policy and an OAuth 2.0 token validator that honors OPA rules plugged into WSO2-IS enabled it to act as the authorization server in the suggested authorization architecture. With the successful achievement at the end of the project, it is proven that any other OAuth 2.0 supporting authorization can also act as this bridge by implementing these 3 components as suitable.

The approach of consuming already established infrastructure based identity in a trusted manner has made Dvaara solution much more extendible while keeping the design simple. It also enables running strict authorization policies at resource access that consider fine-grained details to the level of HTTP methods, resource, and consumer and server identity. Additionally, it also allows dynamic modification of these policies honoring the agile requirements of modern enterprises.

When enterprises look forward to multi-cloud systems, they will be benefited by the approach demonstrated by Dvaara, to control the resource access across the systems in a fine-grained manner under the below mentioned limitations.

6.2. Limitations

While enabling enterprises to make use of multiple clouds, Dvaara also has few limitations on where it can be applied.

- In order for this solution to work as a whole, the workload consumer should identify themselves under the SPIFFE protocol. The SPIFFE implementation used - SPIRE - supports automated node attestation capabilities only for AWS, GCP, Azure, Kubernetes and workload attestation based on Unix, Kubernetes, and Docker. In case we use another cloud provider or if there is a requirement for a different workload attestation mechanism, that needs to be implemented as an extension to SPIRE or we need to use a token-based mechanism with manual intervention.

- The service providing workload needs to be able to work on OAuth 2.0 protocol or at least have the flexibility to place an OAuth 2.0 based access controller in front of them, that can extract necessary details for authorization and honor the decision to deny or permit the request.

The observed response time can also limit the Dvaara usage in an environment where high responsiveness is expected. When Dvaara is to be applied in such an environment, the communication links between the clouds needs to be fast enough and other suggested caching implementations needs to be incorporated to enhance performance.

6.3. Future Work

Dvaara can be improved further in many ways to provide better functionality as follows.

- Dvaara currently makes use of a random string generated by the authorization server as the OAuth 2.0 token. This can be converted into a self-contained JWT token that can carry more useful information between the workloads. If this JWT can be sent signed by the trusted certificate of the authorization server, then the token validation can also be done by the receiving workload agent itself without calling back to the authorization server. Then an external call will happen only for OPA policy evaluation with the details received within the JWT.

- This architecture can be evaluated to expand further with the federation. If there is a trust network between the SPIRE servers, each domain handled by each SPIRE server, can also consume workloads from a totally different domain. In such occasion how this authorization architecture can handle the added complexity can be studied further.

- The solution depends on the TLS layer to securely transport the OAuth 2.0 token from the authorization server to the workload, to make use of it at service call. However, being a bearer token, if some malicious party could get hold of this token, they can make use of this token to access a resource. In that matter, OPA policy can add some protection, but the best solution would be if the token can be bound with the TLS connection initiated by the workload. In that case, it will restrict token to be used by a connection made by the intended bearer of the token, reducing the risk.

- At the token validation, Dvaara expects the workload to make a call to the authorization server and check it's validity. This involves a cost with the call through the network and an OPA policy invocation each time. This can be avoided if the validation response can be cached in the workload for a given time. This has a trade-off between accuracy against the cost as token revocations, policy updates might be neglected for a certain time. Hence this improvement needs to be done with proper cache invalidations in place and cache eviction functionality to avoid infinite memory growth with tokens saved to cache.

- SPIRE server doesn't have a clearly defined way to retrieve its attestation policy used in issuing the SPIFFE IDs. After a discussion in their 'Slack' channel, this is now tracked at [45]. Once this is available it can be integrated into the OPA policy engine replacing current manual policy definition used to identify the clearance level issued to an SPIFFE ID.

- It will also be useful to have an intuitive UI that provides a single view of the authentication and authorization policy across the system. This help administering the system, evaluating current privilege levels of each workload, applying modifications, doing evaluations and auditing.

REFERENCES

[1] R. Chandramouli, M. Iorga, and S. Chokhani, “Cryptographic Key Management Issues and Challenges in Cloud Services,” in *Secure Cloud Computing*, Springer, 2013, pp. 1–30. [Online]. Available: https://link.springer.com/chapter/10.1007/978-1-4614-9278-8_1. [Accessed 23 Nov. 2018].

[2] VansonBourne and Nutanix, Inc., “Nutanix Enterprise Cloud Index,” 2018. [online] Available: https://www.nutanix.com/enterprise-cloud-index/docs/enterprise-cloud-index.pdf. [Accessed 23 Nov. 2018].

[3] A. Jessup, “Building trust between modern distributed systems with SPIFFE,” 21-Feb-2018. [Online]. Available: https://www.slideshare.net/ajessup/building-trust-between-modern-distributed-systems-with-spiffe. [Accessed: 07-Dec-2018].

[4] D. Smith and E. Anderson, “Hype Cycle for Cloud Computing, 2018,” *Gartner, Inc.*, 31-Jul-2018. [Online]. Available: https://www.gartner.com/doc/3884671/hype-cycle-cloud-computing-. [Accessed: 06-Dec-2018].

[5] P. Rabinovich, “Building Identity Into Microservices,” *Gartner, Inc.*, 15-Aug-2017. [Online]. Available: https://www.gartner.com/doc/3784664/building-identity-microservices. [Accessed: 27-Dec-2018].

[6] R. S. Sandhu and P. Samarati, “Access control: principle and practice,” *IEEE Commun. Mag.*, vol. 32, no. 9, pp. 40–48, 1994. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/312842. [Accessed: 27-Dec-2018].

[7] C. De Laat, G. Gross, L. Gommans, J. Vollbrecht, and D. Spence, “Generic AAA architecture,” 2000. [Online]. Available: https://www.rfc-editor.org/info/rfc2903. [Accessed: 27-Dec-2018].

[8] E. Rissanen and Axiomatics, “extensible access control markup language (xacml) version 3.0,” *OASIS standard*, 2013. [Online]. Available: http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-os-en.pdf. [Accessed: 27-Dec-2018].

[9] D. Brossard and Axiomatics, “JSON Profile of XACML 3.0 Version 1.0,” *OASIS standard*, 2017. [Online]. Available: http://docs.oasis-open.org/xacml/xacml-json-http/v1.0/cos01/xacml-json-http-v1.0-cos01.pdf. [Accessed: 26-Dec-2018].

[10] Openpolicyagent.org, "How Does OPA Work?", 2018. [Online]. Available:

https://www.openpolicyagent.org/docs/how-does-opa-work.html.[Accessed: 12-Dec-2018].

[11] Openpolicyagent.org, “Comparison to Other Systems.” in Open Policy Agent Documentation, 2018. [Online]. Available: https://www.openpolicyagent.org/docs/comparison-to-other-systems.html. [Accessed: 27-Dec-2018].

[12] D. Gollmann, “Computer security,” *WIREs Comp Stat*, vol. 2, no. 5, pp. 544–554, Sep. 2010. [Online]. Available: http://wires.wiley.com/WileyCDA/WiresArticle/wisId-WICS106.html. [Accessed: 27-Dec-2018].

[13] R. S. Sandhu, “Lattice-based access control models,” *Computer* , no. 11, pp. 9–19, 1993. [Online]. Available: https://ieeexplore.ieee.org/document/241422. [Accessed: 27-Dec-2018].

[14] D. D. Clark and D. R. Wilson, “A Comparison of Commercial and Military Computer Security Policies,” in *1987 IEEE Symposium on Security and Privacy*, 1987, pp. 184–184. [Online]. Available: https://www.semanticscholar.org/paper/A-Comparison-of-Commercial-and-Military-Computer-Clark-Wilson/1b7dd069a40900cd781a9cf97c8e819c09b4a346. [Accessed: 27-Dec-2018].

[15] D. F. C. Brewer and M. J. Nash, "The Chinese Wall security policy," *Proceedings. 1989 IEEE Symposium on Security and Privacy*, Oakland, CA, USA, 1989, pp. 206-214. doi:10.1109/SECPRI.1989.36295.

[16] Timetoast timelines, “Cloud Computing History timeline,” *Timetoast*. [Online]. Available: https://www.timetoast.com/timelines/cloud-computing-history. [Accessed: 06-Dec-2018].

[17] C. Stamford, “Gartner Forecasts Worldwide Public Cloud Revenue to Grow 17.3 Percent in 2019.” [Online]. Available: https://www.gartner.com/en/newsroom/press-releases/2018-09-12-gartner-forecasts-worldwide-public-cloud-revenue-to-grow-17-percent-in-2019. [Accessed: 28-Jan-2019].

[18] V. D'costa, "Is hyperconverged infrastructure equivalent to the public cloud?", *Web Werks*. 2016 [Online]. Available: http://blog.webwerks.in/cloud-hosting-blog/is-hyperconverged-infrastructure-equivalent-to-the-public-cloud. [Accessed: 08- Jan- 2019]

[19] J. McArthur, K. Yamada, P. Dawson, and J. Palmer, “Magic Quadrant for Hyperconverged Infrastructure,” *Gartner, Inc.*, 02-Jan-2019. [Online]. Available:

https://www.gartner.com/doc/3894101/magic-quadrant-hyperconverged-infrastr ucture. [Accessed: 28-Jan-2019].

[20] M. C. Calzarossa, M. L. Della Vedova, L. Massari, D. Petcu, M. I. M. Tabash, and D. Tessera, "Workloads in the Clouds,", Springer International Publishing, 2016, pp. 525–550. doi:10.1007/978-3-319-30599-8_20.

[21] R. M. Needham and M. D. Schroeder, "Using encryption for authentication in large networks of computers," Xerox, Palo Alto Research Center, 1978. doi:10.1145/359657.359659.

[22] C. Neuman and J. Kohl, "The Kerberos Network Authentication Service (V5)," Sep. 1993. doi:10.17487/RFC1510.

[23] B. C. Neuman and T. Ts'o, "Kerberos: an authentication service for computer networks," *IEEE Commun. Mag.*, vol. 32, no. 9, pp. 33–38, Sep. 1994. doi:10.1109/35.312841.

[24] Amazon Web Services, Inc., "Instance Identity Documents - Amazon Elastic Compute Cloud," Amazon Web Services, Inc. [Online]. Available: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/instance-identity-documents.html. [Accessed: 29-Jan-2019].

[25] Google Cloud, "Verifying the Identity of Instances | Compute Engine Documentation | Google Cloud," *Google Cloud*. [Online]. Available: https://cloud.google.com/compute/docs/instances/verifying-instance-identity. [Accessed: 27-Jan-2019].

[26] N. Sakimura, J. Bradley, M. Jones B. de Medeiros and C. Mortimore, "Final: OpenID Connect Core 1.0 incorporating errata set 1." [Online]. Available: https://openid.net/specs/openid-connect-core-1_0.html. [Accessed: 27-Jan-2019].

[27] D. Hardt, "The OAuth 2.0 authorization framework," IETF. 2012. [Online]. Available: https://tools.ietf.org/pdf/rfc6749.pdf. [Accessed: 06-Dec-2018].

[28] The SPIFFE authors, "SPIFFE – Secure Production Identity Framework for Everyone." [Online]. Available: https://spiffe.io/spiffe/. [Accessed: 29-Jan-2019].

[29] The Istio Team, "Introducing Istio," *Istio*. [Online]. Available: https://istio.io/blog/2017/0.1-announcement/. [Accessed: 06-Dec-2018].

[30] F. Lardinois, "The Istio service mesh hits version 1.0," *TechCrunch*, 31-Jul-2018. [Online]. Available: http://social.techcrunch.com/2018/07/31/the-open-source-istio-service-mesh-for-

microservices-hits-version-1-0/. [Accessed: 06-Dec-2018].

[31] O. Gould, “spiffe support · Issue #1570 · linkerd/linkerd,” *GitHub*. [Online]. Available: https://github.com/linkerd/linkerd/issues/1570. [Accessed: 29-Jan-2019].

[32] Kubernetes - Container Identity Working Group, “ Container Identity Working Group proposal,” *Google Docs*. [Online]. Available: https://docs.google.com/document/d/1bCK-1_Zy2WfsrMBJkdaV72d2hidaxZBhS5YQHAgscPI/edit. [Accessed: 22-Feb-2019].

[33] E. Hammer-Lahav, “The OAuth 1.0 Protocol,” Apr. 2010. [Online]. Available: https://tools.ietf.org/pdf/rfc5849.pdf. [Accessed: 26-Dec-2018].

[34] M. Jones, B. Campbell, J. Bradley and W. Denniss, “OAuth 2.0 Token Binding,” Oct. 2018. [Online]. Available: https://tools.ietf.org/pdf/draft-ietf-oauth-token-binding-08.pdf. [Accessed: 26-Dec-2018].

[35] B. Campbell, J. Bradley, N. Sakimura, and T. Lodderstedt, “OAuth 2.0 Mutual TLS Client Authentication and Certificate Bound Access Tokens,” OAuth Working Group, 2018. [Online]. Available: https://tools.ietf.org/id/draft-ietf-oauth-mtls-07.html. [Accessed: 26-Dec-2018].

[36] Keycloak community and Redhat, Inc., “Keycloak 4.0.0.Final.” Redhat. [Online]. Available: https://www.keycloak.org/docs/4.0/release_notes/index.html. [Accessed: 26-Dec-2018].

[37] M. Schwartz, "Support OAuth MTLS Client Authentication and Certificate Bound Access Tokens · Issue #946 · GluuFederation/oxAuth", *GitHub*, 2018. [Online]. Available: https://github.com/GluuFederation/oxAuth/issues/946. [Accessed: 26-Dec-2018].

[38] D. Jack & V. O. Tom, “OAuth 2.0 Mutual TLS and Certificate Bound Access Tokens in IBM API Connect v5.0.8+,” developer. ibm.com, 28-Sep-2018. [Online]. Available: https://developer.ibm.com/apiconnect/2018/09/28/oauth-2-0-mutual-tls-certificate-bound-access-tokens-ibm-api-connect-v5-0-8/. [Accessed: 26-Dec-2018].

[39] F. Carbone, “How to Enable Open Banking Dynamic Client Registration with Ping Identity.” [Online]. Available: https://www.pingidentity.com/en/company/blog/posts/2018/enable-open-banking-dynamic-client-registration-with-ping-identity.html. [Accessed: 26-Dec-2018].

[40] WSO2 Inc.,“Mutual TLS for OAuth Clients - Identity Server 5.5.0 - WSO2

Documentation.” [Online]. Available:
https://docs.wso2.com/display/IS550/Mutual+TLS+for+OAuth+Clients.
[Accessed: 26-Dec-2018].

[41] A. Cser, “XACML is dead,” *Forrester*, 07-May-2013. [Online]. Available: https://go.forrester.com/blogs/13-05-07-xacml_is_dead/. [Accessed: 17-Feb-2019].

[42] Spiffe community, “spiffe/java-spiffe,” *GitHub*. [Online]. Available: https://github.com/spiffe/java-spiffe. [Accessed: 24-Feb-2019].

[43] P. Jayawardhana, “Dvaara,” *GitHub*. [Online]. Available: https://github.com/Dvaara. [Accessed: 23-Feb-2019].

[44] P. Jayawardhana, “SPIFFE based KeyStore/TrustStore and OAuth2.0 with WSO2 IS Authorization Server,” GitHub. [Online]. Available: https://github.com/Dvaara/demonstration. [Accessed: 23-Feb-2019].

[45] A. Jessup, "Document the registration API · Issue #41 · spiffe/spiffe.io", *GitHub*, 2019. [Online]. Available: https://github.com/spiffe/spiffe.io/issues/41. [Accessed: 23- Feb- 2019]. [Accessed: 23-Feb-2019].

APPENDIX

1. Sample XACML Policy

- Anyone who is trying to access the resource 'MonitoringSystem' should be authorized under this policy.
- 'Any' action of the resource is permitted if the below conditions are met.
    - Organization ID is either WSO2 or Yenlo
    - Nationality LK or EU
    - Work-effort is on Predictions or DetailedView
    - Current time is before 4 pm of the day in +5:00 timezone.

```
<Policy PolicyId="urn:curtiss:ba:taa:taa-1.1"
RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:deny-overri
des">
    <Description>Enterprise Business Authorization</Description>
    <Target>
        <AnyOf>
            <AllOf>
                <Match MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
                    <AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#string">MonitoringSystem</AttributeValue>
                    <AttributeDesignator MustBePresent="true"
Category="urn:oasis:names:tc:xacml:3.0:attribute-category:resource"
AttributeId="urn:curtiss:names:tc:xacml:1.0:resource:Topics"
DataType="http://www.w3.org/2001/XMLSchema#string" />
                </Match>
            </AllOf>
        </AnyOf>
    </Target>
    <Rule Effect="Permit">
        <Description />
        <Target>
            <Actions>
                <Action>
                    <ActionMatch
MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
                        <ActionAttributeDesignator
AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
DataType="http://www.w3.org/2001/XMLSchema#string" />
                        <AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#string">Any</AttributeValue>
                    </ActionMatch>
                </Action>
            </Actions>
        </Target>
        <Condition FunctionId="urn:oasis:names:tc:xacml:1.0:function:and">
            <Apply type="AtLeastMemberOf"
functionId="urn:oasis:names:tc:xacml:1.0:function:string-at-least-one-member-of">
                <Apply functionId="urn:oasis:names:tc:xacml:1.0:function:string-bag">
                    <AttributeValue
```

```
DataType="http://www.w3.org/2001/XMLSchema#string">WSO2</AttributeValue>
                    <AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#string">Yenlo</AttributeValue>
                </Apply>
                <AttributeDesignator
AttributeId="http://schemas.tscp.org/2012-03/claims/OrganizationID"
DataType="http://www.w3.org/2001/XMLSchema#string" />
            </Apply>
            <Apply type="AtLeastMemberOf"
functionId="urn:oasis:names:tc:xacml:1.0:function:string-at-least-one-member-of">
                <Apply functionId="urn:oasis:names:tc:xacml:1.0:function:string-bag">
                    <AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#string">LK</AttributeValue>
                    <AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#string">EU</AttributeValue>
                </Apply>
                <AttributeDesignator
AttributeId="http://schemas.tscp.org/2012-03/claims/Nationality"
DataType="http://www.w3.org/2001/XMLSchema#string" />
            </Apply>
            <Apply type="AtLeastMemberOf"
functionId="urn:oasis:names:tc:xacml:1.0:function:string-at-least-one-member-of">
                <Apply functionId="urn:oasis:names:tc:xacml:1.0:function:string-bag">
                    <AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#string">DetailedView</AttributeValue>
                    <AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#string">Predictions</AttributeValue>
                </Apply>
                <AttributeDesignator
AttributeId="http://schemas.tscp.org/2012-03/claims/Work-Effort"
DataType="http://www.w3.org/2001/XMLSchema#string" />
            </Apply>
            <Apply
FunctionId="urn:oasis:names:tc:xacml:1.0:function:time-less-than-or-equal">
                <Apply
FunctionId="urn:oasis:names:tc:xacml:1.0:function:time-one-and-only">
                    <AttributeDesignator
AttributeId="urn:oasis:names:tc:xacml:1.0:environment:current-time"
Category="urn:oasis:names:tc:xacml:3.0:attribute-category:environment"
DataType="http://www.w3.org/2001/XMLSchema#time" MustBePresent="true" />
                </Apply>
                <AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#time">16:00:00+05:00</AttributeValue>
            </Apply>
            <Apply type="AndFunction"
functionId="urn:oasis:names:tc:xacml:1.0:function:and" />
        </Condition>
    </Rule>
</Policy>
```

2. Sample OPA policy

This below policy in OPA defines the same policy define in XACML previously.

```
package xacml

# input = {
#    user = {"name": "alice",
#            "organization": "WSO2",
#            "nationality": "LK",
#            "work_effort": "Predictions"},
#    resource = {"MonitoringSystem": true},
#    action = {"name": "read"}
#    time =
# }

permit {
   # Check that resource has a "MonitoringSystem" entry
   input.resource["MonitoringSystem"]

   # Check that organization is one of the options (underscore implements "any")
   org_options = ["WSO2", "Yenlo"]
   input.user.organization = org_options[_]

   # Check that nationality is one of the options (underscore implements "any")
   nationality_options = ["LK", "EU"]
   input.user.nationality = nationality_options[_]

   # Check that work_effort is one of the options (underscore implements "any")
   work_options = ["DetailedView", "Predictions"]
   input.user.work_effort = work_options[_]
   #Check the time condition
   time.now_ns() <= 16:00:00+05:00
}
```

3. Sample SPIFFE SVID X.509 certificate

```
Certificate:
   Data:
      Version: 3 (0x2)
      Serial Number: 10608244402538346926 (0x93380e1447d2f9ae)
   Signature Algorithm: ecdsa-with-SHA512
      Issuer: C=US, O=SPIFFE
      Validity
         Not Before: May 13 19:33:47 2018 GMT
         Not After : May 12 19:33:47 2023 GMT
      Subject: C=US, O=SPIFFE
      Subject Public Key Info:
         Public Key Algorithm: id-ecPublicKey
            Public-Key: (384 bit)
            pub:
               04:5a:30:7e:9d:21:92:c4:86:22:ce:76:fc:e3:1b:
               b9:58:60:d9:8f:cd:27:2f:b5:b5:73:7c:df:e3:c5:
               a1:cb:49:9a:ed:8e:e6:08:12:b3:7d:09:2b:80:38:
               2e:23:88:f4:67:ed:3f:b4:31:ff:af:c8:2d:3a:d2:
               cb:ac:8a:6e:33:05:87:a1:ee:2f:6d:50:45:b5:ed:
               6f:8f:a5:ed:e9:67:84:f2:55:f0:70:2b:cb:b3:f9:
               9c:9a:f3:ea:54:af:63
            ASN1 OID: secp384r1
      X509v3 extensions:
         X509v3 Subject Key Identifier:
            87:A5:F3:57:A2:F0:35:AC:C0:F8:64:C4:54:E7:6E:D3:BA:39:C8:E8
         X509v3 Basic Constraints: critical
            CA:TRUE
         X509v3 Key Usage: critical
            Certificate Sign, CRL Sign
         X509v3 Subject Alternative Name:
            URI:spiffe://local
   Signature Algorithm: ecdsa-with-SHA512
       30:64:02:30:13:83:1e:d7:7a:8c:0b:d8:ba:16:4c:74:87:6e:
       b2:d3:d4:19:21:bb:91:a8:0f:69:b8:b8:3d:01:e7:80:03:2a:
       39:b4:1c:d1:97:56:0b:d0:a3:44:a7:4d:95:29:26:09:02:30:
       5d:78:9b:ea:8c:9f:70:5b:9e:4e:1a:3d:49:43:00:c5:0f:b9:
       16:78:40:7a:a0:c9:70:3d:b2:3f:e6:11:18:dd:ac:c9:8b:5e:
       88:d2:e3:75:25:26:13:49:61:92:a9:67
-----BEGIN CERTIFICATE-----
MIIBzDCCAVOgAwIBAgIJAJM4DhRH0vmuMAoGCCqGSM49BAMEMB4xC
zAJBgNVBAYT
```

```
AlVTMQ8wDQYDVQQKDAZTUElGRkUwHhcNMTgwNTEzMTkzMzQ3Whc
NMjMwNTEyMTkz
MzQ3WjAeMQswCQYDVQQGEwJVUzEPMA0GA1UECgwGU1BJRkZFMHY
wEAYHKoZIzj0C
AQYFK4EEACIDYgAEWjB+nSGSxIYiznb84xu5WGDZj80nL7W1c3zf48Why0
ma7Y7m
CBKzfQkrgDguI4j0Z+0/tDH/r8gtOtLLrIpuMwWHoe4vbVBFte1vj6Xt6WeE8lX
w
cCvLs/mcmvPqVK9jo10wWzAdBgNVHQ4EFgQUh6XzV6LwNazA+GTEVOdu
07o5yOgw
DwYDVR0TAQH/BAUwAwEB/zAOBgNVHQ8BAf8EBAMCAQYwGQYDVR
0RBBIwEIYOc3Bp
ZmZlOi8vbG9jYWwwCgYIKoZIzj0EAwQDZwAwZAIwE4Me13qMC9i6Fkx0h
26y09QZ
IbuRqA9puLg9AeeAAyo5tBzRl1YL0KNEp02VKSYJAjBdeJvqjJ9wW55OGj1J
QwDF
D7kWeEB6oMlwPbI/5hEY3azJi16I0uN1JSYTSWGSqWc=
-----END CERTIFICATE-----
```