



Public Key and Multi Factor Based Centralized SSH Authentication System for a Cloud Based Environment Using LDAP

**A dissertation submitted for the Degree of Master of
Science in Information Security**

G.D.D Asanga

**University of Colombo School of Computing
2016**



Declaration

The thesis is my original work and has not been submitted previously for a degree at this or any other university/institute.

To the best of my knowledge it does not contain any material published or written by another person, except as acknowledged in the text.

Students Name: G.D.D. Asanga

Signature:

Date:

This is to certify that this thesis is based on the work of Mr./Ms. under my supervision. The thesis has been prepared according to the format stipulated and is of acceptable standard.

Certified by:

Supervisor Name: Dr. Ajantha Atukorale

Signature:

Date:

Acknowledgement

I would like to thank Dr. Ajantha Atukorale for being my supervisor and his guidance, advice and time throughout the entire project.

Also I wish to thank Dr.Kasun De Soyza, Mr. Kenneth Thilakarathna and Dr. Manjusri Wickramasinghe of University of Colombo School of Computing for encouraging me and for the support provided.

Finally I am thankful to my team lead at WSO2 Lanka (Pvt) Ltd, Mr.Chamith Kumarage and my colleague Ashwin Nallaperuma for their guidance and support.

Abstract

The purpose of this study is to implement a system that provides capability to centrally manage key based SSH authentication and provide access control system with multi factor authentication. This provides multiple users to access multiple servers in a cloud based environment using SSH with enhanced secure access and manageability.

The problem addressed here is to avoid sharing a common private key, password or to avoid using common accounts for remote SSH login. This system helps to centrally manages users, identities and related attributes and also shows how secure it is, in comparison to other systems. This implementation is also focused on a cloud infrastructure which could scale timely.

This thesis discusses the existing systems and compares the advantage of using the proposed implementation and design over other systems that are complex and hard to maintain. The real implementation is focused on a practical scenario for an organization which already has a remote cloud environment where developers and administrators need to access the servers in cloud infrastructure for different purposes. The solution suggested here optimizes the access control mechanism for SSH and also provides easy to manage, centralized, access granting, revocation and tracking capability. This implementation includes a user interface that can be used to easily do all the administrative work and that makes system administrators life easier and avoids repetitive work. This project is mostly based on user management, access control, multi factor authentication and centralizing the administrative work. This study would be much useful to system administrators who are in both networking and systems engineering field.

Table of Contents

Declaration	ii
Acknowledgement.....	iii
Abstract	iv
List of Figures	viii
List of tables.....	viii
Acronyms and Abbreviation	ix
Chapter 1: Introduction	1
1.1 Preamble.....	1
1.2 Brief Overview of the Project	2
1.2.1 Approach	2
1.3 Problem Domain	2
1.4 Motivation	3
1.5 Aims and objectives of the Project.....	3
1.6 Methodology	4
1.7 Scope of the project.....	4
1.8 Overview of the report	5
Chapter 2: Literature Survey and Background.....	6
2.1 Remote Login mechanisms	6
2.2 Standard OpenSSH based access	6
2.2.1 SSH Authentication mechanisms [3]	6
2.2.2 Public Key Authentication in SSH protocol.....	7
2.3 What other authentication protocols available	7
2.3.1 Kerberos	7
2.3.2 Why Kerberos is not used	9
2.3.2 Comparison between Kerberos and Public Key based authentication.	10
2.3.3 OpenLDAP as the identity and the user meta data manager.	10
2.4 Multi-factor authentication.....	11
2.5 The Bastion Concept	11
2.6 Using a cloud based approach.....	14
2.7 Background	15
Chapter 3: Design of Solution	16
3.1 Functional and Nonfunctional Requirements.....	16
3.1.1 Functional Requirements.....	16
3.1.2 Non-Functional Requirements	16
3.2 General Deployment Architecture.....	17
3.2.1 Use internal corporate LDAP server	17

3.2.2 Use external cloud LDAP server.....	18
3.2.3 Use a read replica of internal corporate LDAP server in cloud	18
3.2.4 Multifactor integration	19
3.2.5 User login Flow	19
3.3 Design patterns for a Cloud.....	21
3.3.1 Design for a cloud based environment with a Bastion server.	21
3.3.2 Initial Proof of Concept.....	21
Chapter 4: Implementation.....	23
4.1.1 SSH and LDAP integration	23
4.1.3 Configurations and customizations need to be done in SSH server	23
4.1.4 Fetching the Public key and the group of the user form LDAP	24
4.1.5 What does the script do?	24
4.1.4 Other SSH server configurations.....	25
4.1.5 LDAP Server custom configurations	26
4.1.6 Google authenticator configurations	26
4.2 User interface to manage the system operations	28
4.2.1 User interface	28
4.2.2 Basic functionality of the interface	28
Chapter 5: Results & Evaluation.....	29
5.1 Introduction to the chapter	29
5.1 Test and validation plan	29
5.1.1 Functional Tests	29
5.1.2 Test environment.....	29
5.2 Test Summary and Validation.....	34
5.3 Evaluation	35
5.3.1 Evaluation criteria	35
5.3.2 Evaluated systems	35
5.3.3 Evaluation of each system based on the selected criteria.....	35
5.4 SSH login performance evaluation	37
5.4.1 Test scenario 1	39
5.4.2 Test scenario 2.....	40
5.4.3 Test scenario 3.....	43
5.5 Performance evaluation summary	46
Chapter 6: Conclusion and Future Work.....	47
6.1 Problems Faced	47
6.2 Deviation from original project plan	47
6.3 Future work	47

6.4 Conclusion.....	48
Appendix	49
Appendix I: User Management Interface	49
Appendix II: Output of the LDAP search query	53
Appendix III: LDAP Entry for the user “anura”	54
Appendix IV: SSH Client verbose output	55
Appendix V: LDAP debug Log	56
Appendix VI: SSH Client Log	57
Appendix VII: Creating a new RSA key pair.....	59
Appendix VIII: Adding a user to the LDAP Server	60
1. LDAP schema of an existing user	60
2. Importing a new user to LDAP server.....	60
3. New user is added to the LDAP server	61
4. New user is in LDAP schema	61
Appendix IX: Login as a user to the bastion server	61
1. SSH Client debug output without Private Key provided.....	61
2. SSH Client debug output with Private key provided	62
Appendix X: Testing server behavior when LDAP server is down	63
References	64

List of Figures

Figure 2.3 Kerberos Authentication.....	8
Figure 2.4.1 A typical cloud deployment with a bastion node.....	12
Figure 2.5.1 VPN between corporate datacenter and the cloud.....	13
Figure 3.2.1 SSH authentication with LDAP on corporate data center.....	17
Figure 3.2.2 Bastion and internal servers authenticated via corporate LDAP server.....	17
Figure 3.2.3 SSH authentication with LDAP hosted in cloud.....	18
Figure 3.2.4 Read replica of internal corporate LDAP server in cloud.....	18
Figure 3.2.5 Flow chart for SSH authentication with MFA.....	20
Figure 3.3.1 Standard Cloud setup with a bastion node.....	21
Figure 3.3.2 Initial PoC.....	22
Figure 4.1.6 Scanning the QR code.....	37
Figure 4.1.7 Time based token generated.....	37
Figure 4.2.1 Password based authentication to the application	Appendix II
Figure 4.2.2 Add new users to the system	Appendix II
Figure 4.2.3 Generate QR code for the specific user	Appendix II
Figure 4.2.4 Add/Remove users to/from groups.....	Appendix II
Figure 4.2.5 Delete users from system	Appendix II
Figure 4.2.6 View all the users in the system	Appendix II
Figure 4.2.7 View users in each group.....	Appendix II
Figure 5.1.2 Local scaled down setup for testing.....	30
Figure 5.1.3 LDAP user store.....	32
Figure 5.1.4 Creating a new RSA key pair.....	Appendix VII
Figure 5.1.5 Existing user LDAP schema	Appendix VII
Figure 5.1.6 Adding a new user to LDAP.....	Appendix VIII
Figure 5.1.7 New user added to LDAP successfully.....	Appendix VIII
Figure 5.1.8 New user in LDAP user store.....	Appendix VIII
Figure 5.1.9 LDAP modify user OU.....	33
Figure 5.1.10 LDAP modify user OU command line.....	33
Figure 5.1.11 Change user's OU.....	34
Figure 5.1.12 Auth Failure.....	34
Figure 5.4.1 SSH with password, key, LDAP Password, LDAP key.....	40
Figure 5.4.2.1 SSH with password.....	41
Figure 5.4.2.2 SSH with key.....	41
Figure 5.4.2.3 SSH with Password in LDAP.....	42
Figure 5.4.2.4 SSH with key in LDAP.....	42
Figure 5.4.3.1 SSH with password.....	43
Figure 5.4.3.2 SSH with key.....	44
Figure 5.4.3.3 SSH with Password in LDAP.....	44
Figure 5.4.3.4 SSH with key in LDAP.....	45

List of tables

Table 2.1 Comparison between password and key based authentication.....	7
Table 2.3 Kerberos Notation.....	8

Acronyms and Abbreviation

IAM	Identity and Access Management
SSH	Secure Shell
LDAP	Lightweight Directory Access Protocol
VNC	Virtual Network Control
RDP	Remote Desktop Protocol
SFTP	Secure File Transport Protocol
SCP	Secure Copy
Rsync	Remote sync
X11	X windows System (11 th version)
AES	Advance Encryption Standard
RSA	Ron , Shamir and Adleman (Name of a public key cryptosystem)
PKI	Public Key Infrastructure
NAT	Network Address Translation
MFA	Multi Factor Authentication
ACL	Access Control List
TCP	Transmission Control Protocol
DMZ	Demilitarized Zone
VPN	Virtual Private Network
AWS	Amazon Web Services
IoT	Internet of Things
NIS	Network Information Service
NSS	Name Service Switch
PAM	Pluggable Authentication Modules
VM	Virtual Machine
OU	Organizational Unit
LDIF	LDAP Data Interchange Format
QR	Quick Response
API	Application Program Interface

Chapter 1: Introduction

1.1 Preamble

Identity and access management (IAM) has become an essential requirement and security discipline for almost all nowadays information systems. It expects to manage users, groups, policies, privileges, authentication and authorization within the domain or with deferent federated domains. IAM is crucial to establish well defined security policies within any organization or system. It is essential and beneficial in ensuring security of the assets, cost saving, management control, operational efficiency and business growth. It provides the right access at the right time to the correct person. Organizations have to manage access to resources such as information applications, devices, servers across internal and external platforms. At the same time, providing access and managing number of rapidly growing user identities efficiently without exposing or uncompromising security information is very important. To cater these requirements, use of centralized access and identity management solutions are becoming an immerging trend in most of the organizations. The advantage here is administrators can easily control access to the systems without having to change many properties in different entities or different systems and avoid any repetitive work. Also it helps to track the existing user information, level of access users have and historical information.

In this project the main focus is on centrally governing and managing key based SSH authentication to a large scale, remote, Linux based servers using a centralized authentication management system. The need of implementing centralized key based authentication system emerged because the current, default SSH authentication management methods does not provide such centralized key based authentication solutions. Hence, another user information and identity management system has to be integrated with SSH. One of the most common directory access control systems, LDAP is used in this context to manage users and identities. Additionally, the proposed system is integrated with multifactor authentication mechanism which ensures the secure accessibility for authorized users only and prevents any security breaches due to theft of device or unauthorized access.

1.2 Brief Overview of the Project

1.2.1 Approach

This is an implementation project, in which a working system will be the final outcome compared to a research oriented project. The final outcome of this implementation is not just a prototype and it also provides an enterprise level solution for a real world problem, discussed in section 1.3. In this context, we are specifically focused on implementing production integration for a real world cloud based deployment.

Based on the suggested ideas in the initial project proposal and considering the final expectations, the very first step was to identify the problem ahead correctly and define the scope (discussed in chapter 1, section 1.7). To achieve this, how existing systems works and their limitations were identified at the very beginning. Next step was to follow the available literature on the defined scope (centralized SSH authentication management mechanisms) (discussed in chapter 2). Since this project is more towards to an implementation project, research type literature on the proposed topic was limited. But published papers, articles and definitive guides on underlying core design principles, specifications and protocols were followed in order to identify the optimal and most feasible solution for this implementation. Also suggestions and implementation ideas were mostly found in blog posts and forums. They were well reviewed and then found certain missing points, broken blocks in the flow and some were incomplete stories. Thus some tryouts and different attempts were made to develop the intended solution. Other main objective was to use open source tools to implement the suggested mechanism. Out of the currently available options, OpenSSH was identified as the remote server connection protocol (discussed in chapter 2 2.2). Public key based authentication is used as the authentication mechanism (discussed in chapter 2.2.2). OpenLDAP was identified as the most scalable and reliable open source user information, metadata and identity information manager for this use case (discussed in chapter 2.3.3). Google Authenticator application is also integrated to this system to provide multifactor authentication which provides additional layer of security (discussed in chapter 2.4).

1.3 Problem Domain

There are couple of ways to authenticate SSH (Secure Shell) in Linux based environments. They are discussed in chapter 2. But SSH protocol itself does not provide any built in centralized authentication and management mechanism. Managing large number of users and their identities, credentials is not an easy task for system administrators with default available SSH user management mechanisms. The approach of this implementation is to integrate a centralized authentication and management system which helps to store user credentials and to authenticate authorized users to other systems. The importance of this kind of integration system is the manageability and the scalability. It also provides individual access credentials and avoids sharing common credentials among users. On the other hand a centralized system allows adding, altering and revocation of permissions easily. To provide a manageable centralized authentication mechanism in Linux based environment, system administrators commonly use user stores such as LDAP or ticket granting systems like Kerberos. When we consider above requirements and different SSH authentication types, the more secure and recommended method is public key based authentication (further discussed in chapter 2). This implementation is based on how to develop a workable, scalable, secure, centrally managed Public key based SSH authentication system for a remote cloud based environment. Even with key based authentication, if the keys are compromised or in a theft of device situation, that will create a security issue. That concern is also addressed by providing two factor

authentication to access the remote system. There are multiple challenges such as how this mechanism can be utilized for different cloud based scenarios and how this can be effectively used in an auto scaled, remote server based environments (discussed in detail in chapter 3). Also security is very important and how to achieve security by following different approaches is concerned here.

1.4 Motivation

The main motivation behind this project is to find a solution to a common problem by integrating existing resources. It also needs new feature development work and research work for analyzing the feasibility of the integration. After the prototype is successfully tested on a test environment, it is supposed to be practically implemented. The practically applicable environment for this project implementation is selected as my current work place, WSO2 Lanka (Pvt) Ltd. Hence there is a good motivation factor where, upon success of the research and implementation, it will be realistically used for day to day activities as a solution to the previously mentioned problem domain.

The other major motivation is addressing a problem that is very important for a cloud based environment. Now the modern trends and design principles of most IT systems takes the cloud based approach and the concept of on premise data center is getting obsoleted. Due to many reasons (discussed in chapter 2.6) most of the organizations and institutes tend to use cloud based systems for their IT related operations and still the security and accessibility has to be addressed specifically within these types of deployment solutions.

1.5 Aims and objectives of the Project

The primary objective of this project (as stated in the project proposal) is to implement a secure, manageable and scalable SSH authentication mechanism to remote servers for a cloud based environment. That includes using a user and key management features where identities are managed centrally.

The key deliverables of this project are listed below.

- Store and manage all user specific information (user name, public key, group etc.) in a directory service (In this case, OpenLDAP is used).
- Integrate remote Linux servers to work and use data stored in LDAP server for authentication.
- Create user home, shell, and other information upon initial login.
- Revoke, edit update user permission as per the requirements.
- Manage and identify the users and their level of access.
- Enable multifactor authentication.
- Design a user interface to easily manage all the administrative tasks.

1.6 Methodology

Initial research and readings were done on, how a directory service server can be used as the public key holder while a remote SSH server checks a specific user's validity by verifying the information provided by the user or the client program at the login. It focuses on what attributes, characteristics and capabilities of OpenSSH and OpenLDAP integration can be used to accomplish the expected goals. The main approach here is to find out how the SSH key based authentication can be integrated with a directory service such as OpenLDAP which can manage users, attributes and public key etc. That gives the capability to build a centrally managed identity management system. Then during authentication process, remote server can verify user's authenticity against the information stored in the directory service. That is the authentication part.

Next part is the authorization. That can be done by validating other metadata in the directory service. This will enable different access levels to different resources. This approach provides a huge advantage that, no user related keys or information need to be pre stored in the remote server because they are managed within the directory service integrated with SSH. This is a major requirement in auto scaling environments.

Next is to develop an interface to manage existing users in LDAP to add to the SSH allowed user groups etc. It should allow all administrative actions such as adding, removing and editing users. That interface also makes it easy to track details of users operated within the system.

1.7 Scope of the project

In this implementation LDAP is used as the centralized directory information service to manage and store identities. The well-known Kerberos protocol is not selected here due to many reasons and that is explained in chapter 2.3.2. The addressed scope is related to a real world industrial problem. Finding an optimized, manageable, and scalable secure solution is the main objective. That becomes this implementation more related to a practical implementation than an evaluation prototype. Reviews on manageability and security will be carried out for the proposed method. Also it will be discussed how secure the suggested approach, what security vulnerabilities can rise and elaborate how the manageability becomes easier with the proposed system. Also performance evaluation will be done to compare the proposed system with other systems. The deployment is designed and aligned towards a cloud based approach. In this entire project, OpenSSH is used as the SSH protocol. OpenSSH is an open source protocol which provides encryption for network related operations like remote login and remote file copying [1]. The defined scope includes enabling multifactor authentication to enhance the security. OpenLDAP is used as the central user and identity management service.

1.8 Overview of the report

Chapter 2 (Literature Survey and the Background) provides the literature related to the project and the overall research methodology used in the project. There a compare and contrast of different technologies and protocols (SSH, LDAP, Kerberos and different authentication methods) associated with the core implementation of the project is carried out and given facts in an analytical manner why the proposed solution is suitable and the practical approach of it.

Chapter 3 (Design of Solution) describes the design principles followed for this implementation and the advantages of them. It specifically focuses on a cloud based approach and how each component (LDAP and Public Key, Google Authenticator etc) is integrated to provide the final outcome is explained there.

Chapter 4 (Implementation) is based on the configurations and integrations done for the project. It shows the customizations, scripts used and basic functional operations and how they work.

Chapter 5 (Results and Evaluation) show what are the test plans carried out to test and validate the implemented system and based on the test results, a proper evaluation and a validation is done. This includes functional tests can performance tests. The evaluation is based on different criteria and different reference systems have been used for evaluating the proposed system.

Chapter 6 (Conclusion and Future Work) is a recap of the proposed solution and how it was achieved. It describes up to which extent the deliverables and outcomes were completed and what will be done as future work to improve the quality and the security of the project.

Chapter 2: Literature Survey and Background

2.1 Remote Login mechanisms

With the origin of networking concepts and networked systems, requirement of remote access and communication systems became essential to access them remotely and work remotely with those services. People could work from different geographical locations without being need to be physically present near the device by using remote login mechanisms. Then, with the growth of technology, requirement of security too came up. That emerged lots of concerns regarding identity management and securing assets of any organization. Different protocols and practices were introduced to cater those requirements. There are some common remote access protocols widely used by developers and administrators [2].

Remote CLI tools

- Telnet
- rlogin
- SSH

Remote Desktop

- VNC
- Microsoft RDP
- Citrix

In this context the focus is on most commonly used remote login mechanism, SSH which is highly associated with Linux based environments. The easiest and most efficient way to access the remote Linux based servers is to use SSH. The protocol SSH (Secure Shell) creates an end to end encrypted channel for logging into a computer in a network and allows running commands on a remote computer. More details on SSH is discussed below.

2.2 Standard OpenSSH based access

With the growth of the security awareness, for most operations engineers and developers tend to use SSH protocol as secure and easy to access mechanism with encryption, for their remote server communications. SSH protocol can be used to provide encryption through tunneling for the protocols that do not provide encryption by themselves. SSH is also used in its associated protocols like SCP, SFTP, rsync etc. Basic introduction to SSH authentication mechanisms are given below. OpenSSH is selected as the protocol to establish the remote connectivity since it is completely open source and use strong cryptographic algorithms (AES, RSA etc). Also it provides X11 forwarding, port forwarding, agent forwarding, strong authentication mechanisms. Also it can be integrated with additional extension such as Google Authenticator etc. These features are essential in the requirements of this project.

2.2.1 SSH Authentication mechanisms [3]

- Password based authentication
- Public key based authentication
- Host based authentication
- None (not recommended)

Each of them has both advantages and disadvantages. We can differentiate the two most common authentication methods Password based and Key based authentication as given in the table 2.1 [4],[5],[6].

Password based authentication	Key based authentication
To make the password more secure, it has to be long and composed of random characters (upper/lower case), numbers and symbols. But those types of Passwords are difficult to be remembered and can be easily forgotten. Then it has to be written down somewhere, which is also not secure.	Don't have to be remembered. But has to be protected. (This can be done using a passphrase). Still passphrase has to be either remembered or write down somewhere. Can be secured using multifactor authentication which provides resistance to theft or compromise of the private key.
Weak passwords can be guessed or cracked (using dictionary attack or brute-force attack)	SSH keys are generated using RSA algorithm and highly unlikely to crack since it is a unique string.
Even SSH sends the password in a secured channel, can be captured if the remote host is compromised (fake servers may be providing bogus logins) [7]	In key based authentication, private key remains within the client and no sensitive value is sent to the server. PKI creates a secure mechanism for session key exchange.
A strong password that can be remembered is good. Hence a remembered strong password is highly unlikely to leak.	Key files are stored in client machine and if not protected by a passphrase compromised or stolen client machines can be really vulnerable.

Table 2.1 Comparison between password and key based authentication

Based on above facts it can be concluded that the more secure SSH authentication method is using keys than using passwords. That is another main reason why Public key based SSH authentication is selected in this project.

2.2.2 Public Key Authentication in SSH protocol

Public key authentication is based on asymmetric key encryption. The client will generate a pair of keys known as public and its corresponding private key. The private key is only known to the client and the public key must be known and authorized by the server. When client wants to connect a specific user in a remote server, it proves to the server that it has the corresponding private key that matches the public key authorized and known by the server [8]. If the verification is success, user will be authenticated.

2.3 What other authentication protocols available

2.3.1 Kerberos

Kerberos is an authentication system implemented by using a ticket granting mechanism for authenticated users. It was originally developed by MIT for the project of Athena [8][9]. Kerberos follows the symmetric encryption and trusted third-party concept. It is different from the public key infrastructure model. But the latest version (v5) is improved to work with asymmetric encryption functions. The main objective of Kerberos is not to send the password through the network, instead it uses the password hash check from both side of the connection and password itself is used to encrypt, decrypt (symmetric) the encrypted password hash.

Normally this process is called Kerberos ticket granting process. The ticket granting concept can be explained as follow. Table 2.3 provides notation related to the figure 2.3

Notation	
Client Machine	C
Services Server	S
Key Distribution Server	KDC
Private Key for “ x ”	K_x
Session Key for the Client	$K_{S,c}$
Session key for the Services Server	$K_{S,s}$
Client IP	IP

Table 2.3 Kerberos Notation

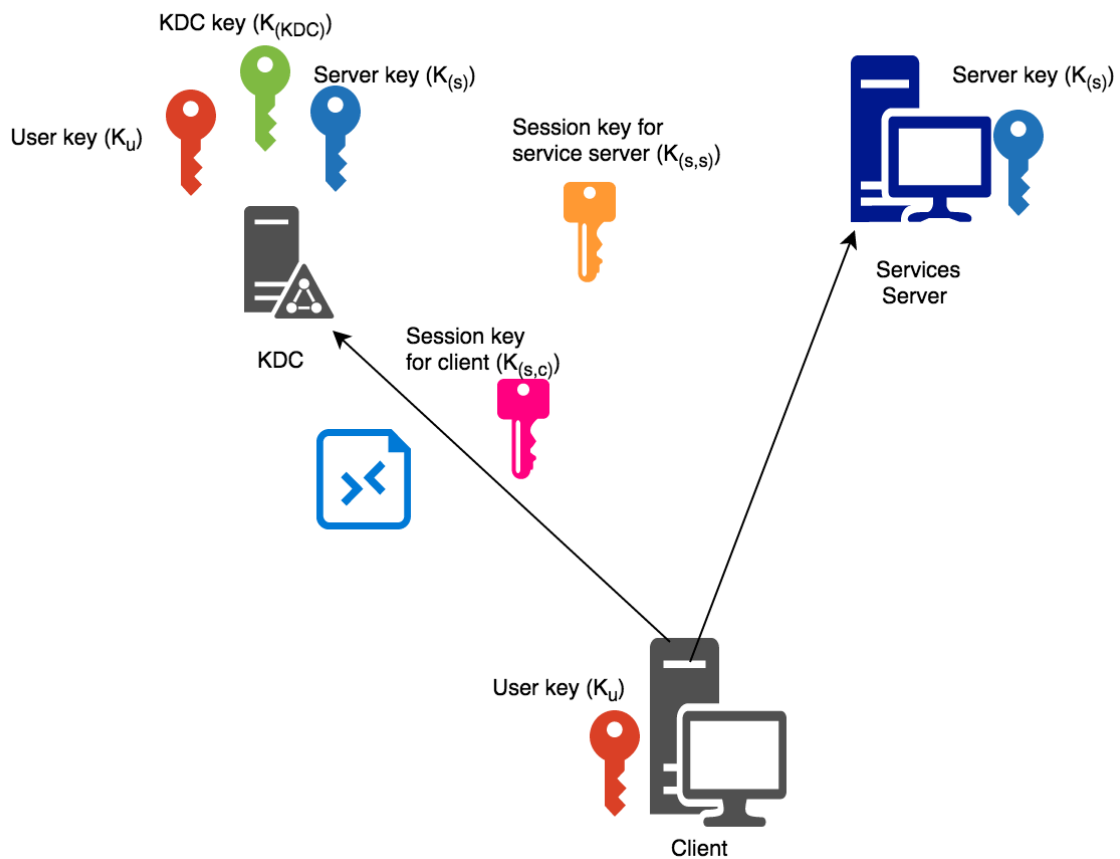


Figure 2.3 Kerberos Authentication [10]

The three main things in Kerberos is Kerberos client, Kerberized services and the KDC (Key Distribution Service). The KDC maintains all the passwords in a single location. When a user tries to authenticate, it doesn't send password in clear text.

Kerberos contain principles made of users and services. The KDS stores a secret encryption key for each principal. The keys are created by system administrators and they are only shared within the principle and the KDC. If a packet comes encrypted using this key, then principle can trust that it came from the KDC. KDC's private key is used to assure the integrity of the authentication. In the diagram depicted in figure 2.3, blue keys are used for users and red for

the services key and KDC's key is given in green. It is obvious that a packet encrypted by KDC with user key can be decrypted by the user. The use of this key is to share a session key which is not permanent within user and KDC. The significant fact that is used here is anything encrypted that comes from KDC is trusted by the principle. The important thing is, each principle trusts that anything encrypted with its private key comes from KDC. Any encrypted packet that comes from user which is encrypted by services private key is accepted as an authenticated packet since only KDC can send that to user.

When a user logs into a device with user name and password, a packet will be transmitted to KDC requesting a ticket granting ticket (TGT). You need a valid ticket to access a resource because it proves your identity. The user principle will then start to use this TGT to receive tickets for all of the services it wants to access. But to avoid a stolen key being misused, these tickets do have an expiry time. KDC can check user's validity by checking him in its user principle database and then create a TGT. This contains the users name, the KDC name, the client IP address and the new key called the session key. Then KDC encrypts all these information with his private key and send to the agent. This cannot be read by the client. The KDC then sends a packet that contains TGT and another copy of the session key. This is encrypted with the users' private key. The client uses his private key (which is the password typed in when he logged into the machine).

Now client generates an authenticator which contains his user name, IP address and the timestamp. This authenticator is encrypted by the session key it just received from KDC. Then the client generates a packet which includes the unopened TGT, the authenticator and the name of the requested service and sends the packet to the KDC. Then KDC decrypt the TGT, and that proves it hasn't been opened. KDC can use the TGT to check the info on the authenticator. By this time KDC can trust the identity of the user where KDC sends a service ticket to the client. The service ticket is a time stamped one and it is valid only for a given period of time and valid only for that specific session only. This ticket contains the user name, the service name, the IP of the client computer and the new session key. Next KDC uses the services' private key and encrypt this ticket. Client cannot open or modify this encrypted ticket because he doesn't have the services' private key. KDC now put this ticket and the new session key and encrypt that with the previous session key and sends to the client. Client then decrypts the packet, gets the new session and encrypt a new authenticator and sends it to the service. Service can use its private key to decrypt the service ticket. Upon successful decryption the service can confirm that this must have come from the KDC because he is the only one to have the services' key. Service can then open the authenticator using the session key and get the information is the authenticator to verify. Upon successful verification the service sends back an acknowledgement to the user and the user is authenticated to the service.

2.3.2 Why Kerberos is not used

As discussed above, Kerberos is based on passwords and it was critically analyzed and shown in chapter 2.2.1 that the password based authentication is insecure because password can be guessed and due to other vulnerabilities associated with passwords like dictionary attack or brute force attacks. In the paper [11] it has discussed in detail that there are known vulnerabilities and weaknesses in Kerberos. Some of them are as listed below.

- Replay attack
- Secure Time Services needed
- Password guessing attacks
- Spoofing login
- Inter-Session Chosen Plaintext Attacks

At the same time if the Kerberos server gets compromised, then that will fail the entire security infrastructure related to access. To avoid that, Kerberos (as the centralized authentication server) has to be managed with intensive care, high security and high availability. So even though Kerberos can be used for this requirement, which provides authentication and integration with SSH, there are some other disadvantages compared to LDAP and hence Kerberos is not evaluated in this implementation. Detailed analysis is done on why LDAP based public key authentication is better than the Kerberos TGT mechanism correspond to the selected scope.

2.3.2 Comparison between Kerberos and Public Key based authentication.

We have already discussed about the public key infrastructure and how that works. But to authenticate user to the server, Kerberos protocol can also be used. Kerberos also provides centralized authentication and manageability features. Also in literature referred [11] it was found that *“Kerberos is not effective against password guessing attacks; if a user chooses a poor password, then an attacker guessing that password can impersonate the user. Similarly, Kerberos requires a trusted path through which passwords are entered. If the user enters a password to a program that has already been modified by an attacker (a Trojan horse), or if the path between the user and the initial authentication program can be monitored, then an attacker may obtain sufficient information to impersonate the user. Kerberos can be combined with other techniques, as described later, to address these limitations”*. Some of the other challenges in using Kerberos are given below.

- Firewall and NAT issues.
- Number of Ports need to be opened to configure Kerberos.
- Configuration issues (many configurations, need special knowledge).
- Dependent on time service.

In summary, it can be concluded that the security of Kerberos depends critically on synchronized clocks. Also Kerberos protocols mandate mutual trust among the client, server, authentication server and time server which has to be well secured to assure the reliability of the Kerberos based authentication.

2.3.3 OpenLDAP as the identity and the user meta data manager.

OpenLDAP is an open source version of lightweight directory access protocol which is mostly used with Linux based environments. Some of its native features are given below.

- IT can be used to store people and other organizational attributes.
- It helps to manage different resources and in this context it is mainly used as identity manager.
- It can be integrated with many other programs.
- Since LDAP is a centralized information holder, it's easy to keep all user data in one storage than maintaining many.
- It helps to easily add remove edit different users and attributes and also group them, which provides much more manageability.
- Many other programs contain libraries that can be integrated with OpenLDAP which makes integration easy.
- It is highly read optimized and that provides efficient read access to other systems to the LDAP server [12],[13].

2.4 Multi-factor authentication

Multifactor authentication (MFA) is a commonly used security best practice related to most of the modern systems which provides additional layer of security. It could be something user has with him (authenticator app or security token), something user knows (password or security question) or something that is uniquely identified for the exact user (fingerprint or voice). In most IT systems, token based MFA is mostly used with username and password authentication. When token based multifactor authentication is enabled, an authentication code is prompted as soon as the user provides the correct username and password. Each individual user has to have a separate token and they usually generated by a token generator. These token generators comes in both virtual and hardware format. The well-known RSA SecureID® [14], RSA SecureID Software token and Google authenticator [15] are couple of examples. Both hardware and software tokens are used and using software token is much easier because it comes as a mobile app or desktop application.

In this implementation, Google authenticator app is used as an additional layer of security other than key based authentication. The main advantage is even though an unauthorized user gets access to the private key of the user or if the LDAP server gets compromised, still there is an authentication barrier that the malicious user has to pass. Since this is mostly included in a mobile device, getting access to the mobile device is again difficult for the attacker. This is integrated to the system by making the remote bastion server MFA aware. SSH server need to be configured with libpam-google-authenticator installed [16]. The token can be configured either time-based or sequential-based. In this context, time-based token method is used and a random code is generated timely.

2.5 The Bastion Concept

When it comes to providing access to a remote deployment, it is the best practice of any system administrator not to open the application servers directly to internet for people to access. He should expose only the ports required and others ports should be closed by default to avoid security breaches or attempts. One of the best approaches to provide secure SSH access to remote servers is to use a bastion host. Bastions hosts are setup in the public subnets and they are typically accessed using SSH or RDP. It can be used as a "jump" server which allows SSH or RDP to other internal servers. Once this is configured with proper security standards (listed below), it works as a bridge/gateway to the internal private subnet [17],[18].

- Setup firewall rules and network ACL and open only the required ports and allow only from the specific hosts (Only from your organization is recommended).
- It is strictly advised not to store SSH private keys in bastion hosts and keep them securely in user machines only.
- Instead of storing private keys in bastion host use SSH agent forwarding to connect to nodes in the private subnet by using bastion as a jump box.
- Harden the operating system as required and avoid running any other applications or services on bastion host to avoid both potential internet and intranet based attacks.
- If there are services run by default or any port open by default and that is not required for the base functionality of the bastion host, stop them and deactivate from Linux service level to avoid them starting back upon a server restart. This will help to minimize any potential attacks from outside and exploiting zero day attacks.
- Setup SSH-Agent forwarding (For Linux) or Remote Desktop Gateway (Windows) [18]

- It is advisable always have more than one bastion host (A high available setup) that is reliable for possible disruptions because bastion is the only host you have to access the remote environment.
- The firewall should only accept SSH protocol (TCP port 22) in specific to this case study and the instances also should only accept SSH and that is only from bastion host.

Also, if a VPN is not used, we need to have a bastion server to secure the environment. Otherwise all the applications servers has to have a public IP (assuming the application servers are setup in a remote cloud) to connect to them remotely. Exposing application servers to public internet is highly insecure. Hence the easiest way is to use a bastion host which is deployed in the demilitarized zone (DMZ) and allow users to access the internal application servers through the bastion host. Thus bastion server concept is mostly used in cloud environments.

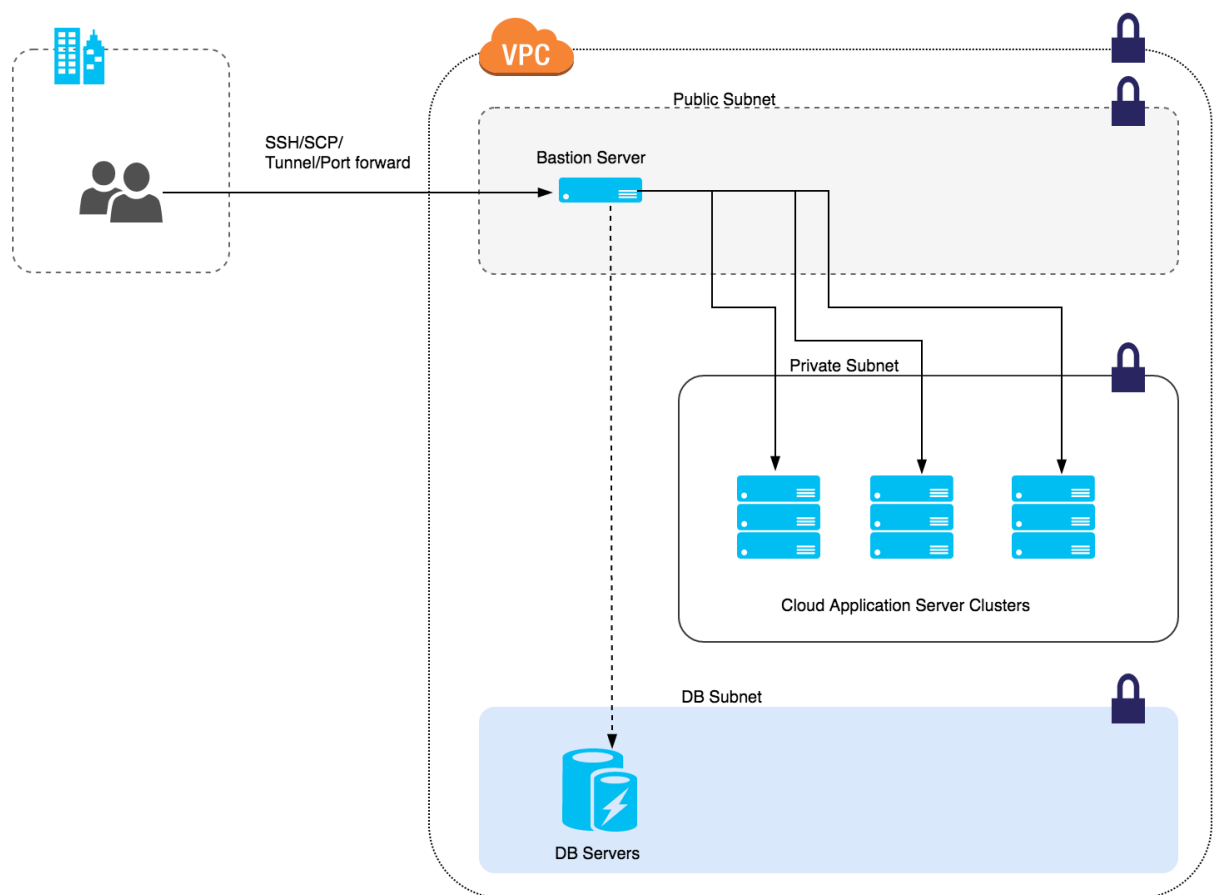


Figure 2.4.1 A typical cloud deployment with a bastion node

Figure 2.4.1 shows generic cloud deployment network architecture. This is the deployment pattern that is used as the example environment for the implementation in the project. This deployment pattern uses a bastion server as the intermediate device that connects the user to any cloud server. Application deployment network is isolated and secured by making it a private subnet and SSH access (in this context we do not discuss other access requirements) is provided to the application hosting nodes through a bastion servers [18].

In order to provide additional security, there are many advantages in using a direct VPN connection to remote cloud servers. A typical VPN connection looks below (figure 2.5.1).

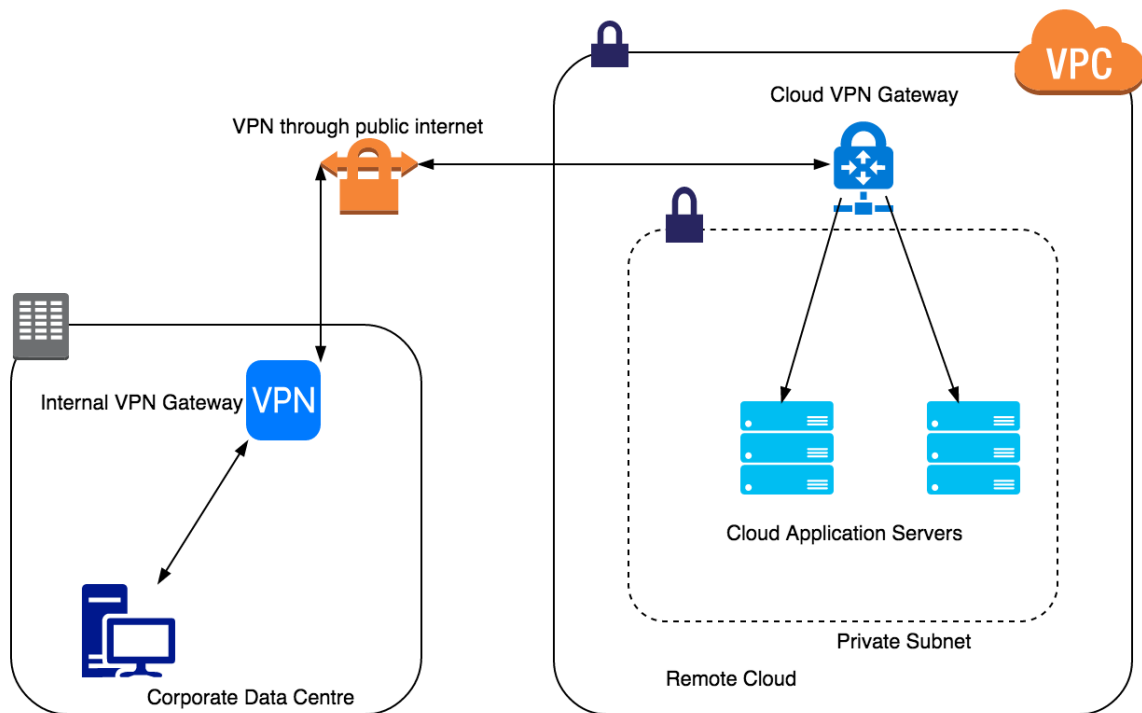


Figure 2.5.1 VPN between corporate datacenter and the cloud

A VPN connection provides end to end secure tunnel and normally established between the organization's corporate data center and the remote cloud. Since VPN is a secure private channel, there could be an argument that a bastion host is not required and users can directly SSH to the remote environment. This has both advantages and disadvantages.

Advantages

- No bastion host is needed and setup and maintenance effort for a bastion host is not required.
- Users can directly connect and also copy files and create tunnels to remote servers.
- No need to do SSH forward agent.

Disadvantages

- If no bastion is used then remote application server's SSH port (22) has to be opened at least to the corporate data center through the VPN and that could lead to potential try out of attacks.
- It is going to be very difficult to maintain server IP addresses in client side and everyone has to know the remote sever IP addresses and if they are changed, everyone who connects to the remote environment has to alter the IP addresses in their configuration files (usually in the `~/.ssh/config` file) each time a new server starts or an existing server get terminated. But if a bastion server is used, host aliases has to be maintained only there and that is lot more manageable than changing configurations within lots of users.

These issues can be easily addresses in an auto scaled environment by using a bastion host in this context. Because in a fully automated, high available clustered environment, servers are

terminated and new servers are started regularly and the new servers usually gets a different IP address. If a single bastion host is used then it is much easy to automate the host aliases in bastion to get automatically updated using scripts than they are updated in each client's machine. Hence clients (simply referred to the remote user machines) only need to know the bastion hosts SSH configuration information and, bastion will have all the private subnet hosted information within its SSH configurations. That is another aspect addressed in this project. So using a bastion host with a VPN is much more secure and that avoids opening bastion host directly to the public internet.

2.6 Using a cloud based approach

Even computer and information related technologies grew up rapidly during past few decades, most of the IT solutions were deployed in-house data centers and they were managed by the same organization. But with the immerging cloud based technologies, this growth became exponential and many organizations, institutes moved fully or partially (cloud based extended data center concept) to cloud based technologies. Not only that instead of their own managed systems, third party managed solutions became much popular and outsourced projects with different scales became more manageable and profitable with support of more professional business IT solution providers.

“Traditionally organizations have looked to the public cloud for cost savings, or to augment private data center capacity. However, organizations are now primarily looking to the public cloud for security, realizing that providers can invest more in people and processes to deliver secure infrastructure.” Google Security Whitepaper [20]

With these new cloud technologies, access and identity management, authentication, authorization became much more concerned factor and at the same time they needed to be scalable and manageable too. Because earlier most servers and systems were managed by a separate person or a role commonly known as systems administrator and others were not allowed to connect to server back ends. But modern agile development methodologies involve not only systems administrators, but also developers and some other roles in the development lifecycle need to have access to the systems and servers to a certain level with control. Early days physical servers, server racks were used and they had permanent IP address assignment in operations level. But in modern cloud based systems, the presence of a physical server is replaced by virtual machines or containers which make an IP address is not distinguished anymore. It could change at any given moment with the features like auto scaling. The ultimate requirement is users should seamlessly be able to connect to them at any given moment for different backend operations.

2.7 Background

The initial idea for this project came related to a requirement for the remote access for the author's organization, WSO2 (Pvt) Ltd, which is a middleware company which provides open source middleware solutions. Currently WSO2 provides a cloud solution for API management, identity management, integration and device management (IoT). This cloud is managed by WSO2 Operations team. This public cloud contains more than three hundred remote servers hosted in Amazon Web Services (AWS). The production environment and its staging environment is regularly accessed by Developers, Dev-Ops for development work, debugging, push artifacts, configuration changes, apply patches and upgrades. It's not just only connecting over SSH, developers and operation team will be using other associated protocols such as SSH tunnels, SCP and r-sync for cloud operations. Hence there has to have a proper access control mechanism for system administrators to grant and revoke SSH access to the developers time to time. Not only that, it shouldn't be a difficult process and it should be scalable at the same time. Because in a cloud based environment scalability is very important and servers could get terminated and new servers could start at any moment.

Currently system administrators use key pairs and provide users with relevant information and users use those keys or passwords to authenticate to a remote server. This is the current scenario used by WSO2 in the public cloud context to authenticate developers to the remote WSO2 Public Cloud servers. SSH access to the servers in public cloud is only allowed to the WSO2 employees (Developers, System administrators etc). System administrators have to provide passwords or add user's public keys to the remote servers and remove them time to time. This process is cumbersome and managing users (adding new user, changing existing user, revocation of permission) is difficult and not tracked properly. Also administrators sometimes have to create and share common keys among users and that cause the accountability issues on who access the system and unable to trace. Another issue is, in an environment where servers auto scale, existing servers may be terminated and new servers get started. So system administrators have to configure each new server with access information for each user. This implementation addresses that issue and provide a simplified solution.

Chapter 3: Design of Solution

3.1 Functional and Nonfunctional Requirements

In this chapter, the core concept of the implementation is discussed with proper diagrams and relevant explanations. The principles and concepts used in this chapter were discussed in detailed in chapter 2. More readings in detail can be found in the relevant reference materials. The major requirements and good to have features are discussed here.

3.1.1 Functional Requirements

Functional requirements of this project can be identified as below.

- Centrally manage user identities within LDAP server. This is one of the major requirements as discussed earlier and suggested implementation will use LDAP as the central identity management service and it will act as the identity and users attributes storage such as username, home, login shell and public key.
- Use Public key based SSH authentication. This approach use the most secure SSH authentication mechanism.
- Enable multifactor authentication at user creation to provide an additional security layer.
- Avoid storing any user specific configuration on remote SSH nodes and create user attributes etc. upon initial login to support auto scaling capabilities.
- Add, remove, update, and revoke users and corresponding privileges centrally.

3.1.2 Non-Functional Requirements

Non-functional but optional requirements can be identified as below.

- To be able to work with a jump box (bastion) use case and all the authentications should happen through the key based login.
- User friendly interface to add, remove, update, revoke users and corresponding privileges.
- Secure connectivity between the server and the LDAP server.

3.2 General Deployment Architecture

3.2.1 Use internal corporate LDAP server

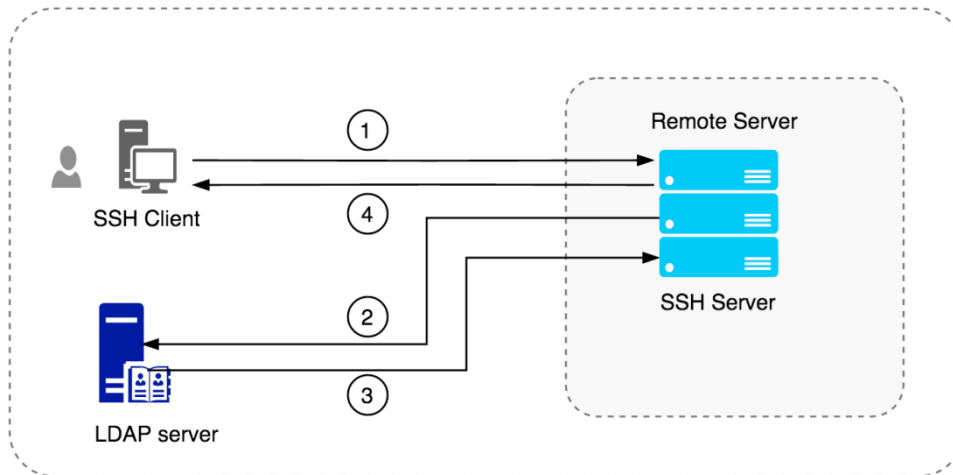


Figure 3.2.1 SSH authentication with LDAP on corporate data center

The setup on figure 3.2.1 shows how the client machine, remote cloud server and LDAP server can be arranged. In this design pattern, LDAP server is hosted in corporate data center and this will need remote server to be able to connect to the corporate data center hosted LDAP device. This has some security issues. Because this design pattern allows cloud environment initiated connection to be established with the internal data center. If this happens through public internet directly, that is not a well-accepted security best practice. If this kind of setup is used, it is always advisable to create an end-to-end VPN with client data center and the cloud. Also this is not recommended in a setup like in figure 3.2.2 where when there is a bastion server in the setup. In that kind of setup the purpose of using a bastion host in DMZ subnet and applications in an internal private subnet is to tighten the security of the environment. So we have to move the LDAP server into the cloud environment. That is depicted in figure 3.2.3.

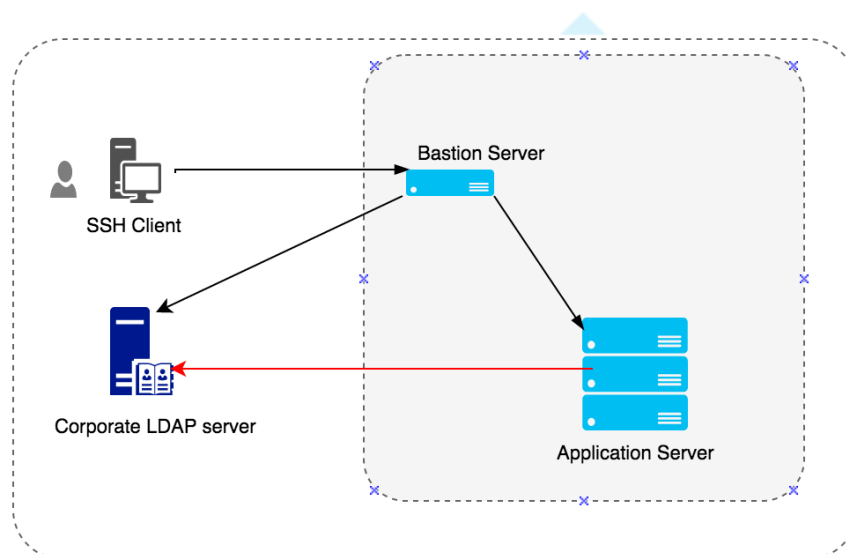


Figure 3.2.2 Bastion and internal servers authenticated via corporate LDAP server

3.2.2 Use external cloud LDAP server

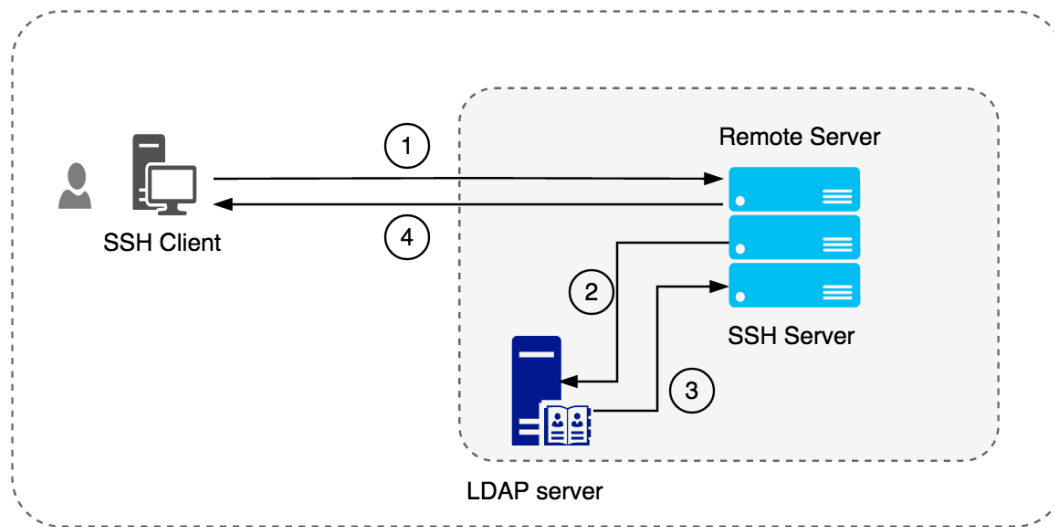


Figure 3.2.3 SSH authentication with LDAP hosted in cloud

According to this setup the LDAP server is hosted in the remote cloud itself and no need to open connections from outside other than for SSH protocol communication. The main concern here is to secure the LDAP server and manage. Normally most organizations keep a read replica of the directory server in remote cloud if the same directory server is used internally and externally for identity management. So that setup is depicted in figure 3.2.4. Since this is a read replica, that assures no one writes to that other than the master.

3.2.3 Use a read replica of internal corporate LDAP server in cloud

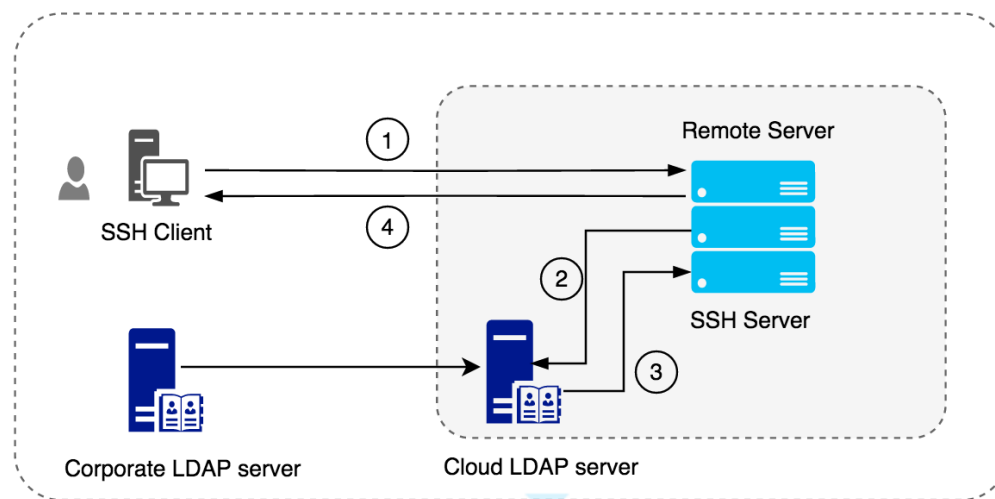


Figure 3.2.4 Read replica of internal corporate LDAP server in cloud

The figure 3.2.4 depicts the suggested implementation use case in its simplest form. The main components are SSH Client, SSH server and the LDAP server. The communication channels taken place in the login process is listed below.

1. OpenSSH client initiate the login request
2. OpenSSH server query the LDAP server for matching users public key
3. LDAP provides the public key to the OpenSSH server
4. Upon validation, login is accepted or rejected

Step 1 and 4 are the standard procedures while this implementation focuses more on step 2 and 3. In the default SSH authentication, once the SSH agent initiate the authentication request, client and server need to discuss the authentication specifications. First client and server agree on the SSH protocol versions and algorithms. Then the server presents the server's host key (only first time) and if it is for the first time login attempt, client has to accept the host key and if not, the client verifies the identity of the remote host. Then client and server agree on the Key exchange protocol. And out of available authentication methods it selects the public key. Then the SSH RSA key validation happens checking with the available public keys in the authorized key file. Once this is completed, a session key is generated. Then the secure channel is established.

Above is the standard SSH key based authentication procedure. But in this implementation we avoid storing the public key at the remote host's authorized_key file and the approach is to query it from the LDAP server. How this can be done is one of the research approaches in this implementation. The other research aspect is storing the user's public key associated to a user attribute in the LDAP server. When a user is trying to SSH, based on the user name the SSH server is expected to pick the public key from the LDAP server querying the user by his user id. Then the standard SSH authorization happens.

3.2.4 Multifactor integration

For multifactor authentication, Google authenticator application is used and user needs to provide the correct token (six digit code) when authenticating to the bastion server. At initial user creation a quick response code is generated using Google authenticator and user needs to scan the code from his mobile app and store the token. These token details are then stored in the LDAP server per each user. Upon user creation, the token information configurations are written to the user's configuration file in his home directory. This will ensure user is able to login to the bastion server with the related token he observed when the account is created.

3.2.5 User login Flow

The diagram given below (Figure 3.2.5) clearly shows the flow of authenticating a user to bastion and from there to an internal private server.

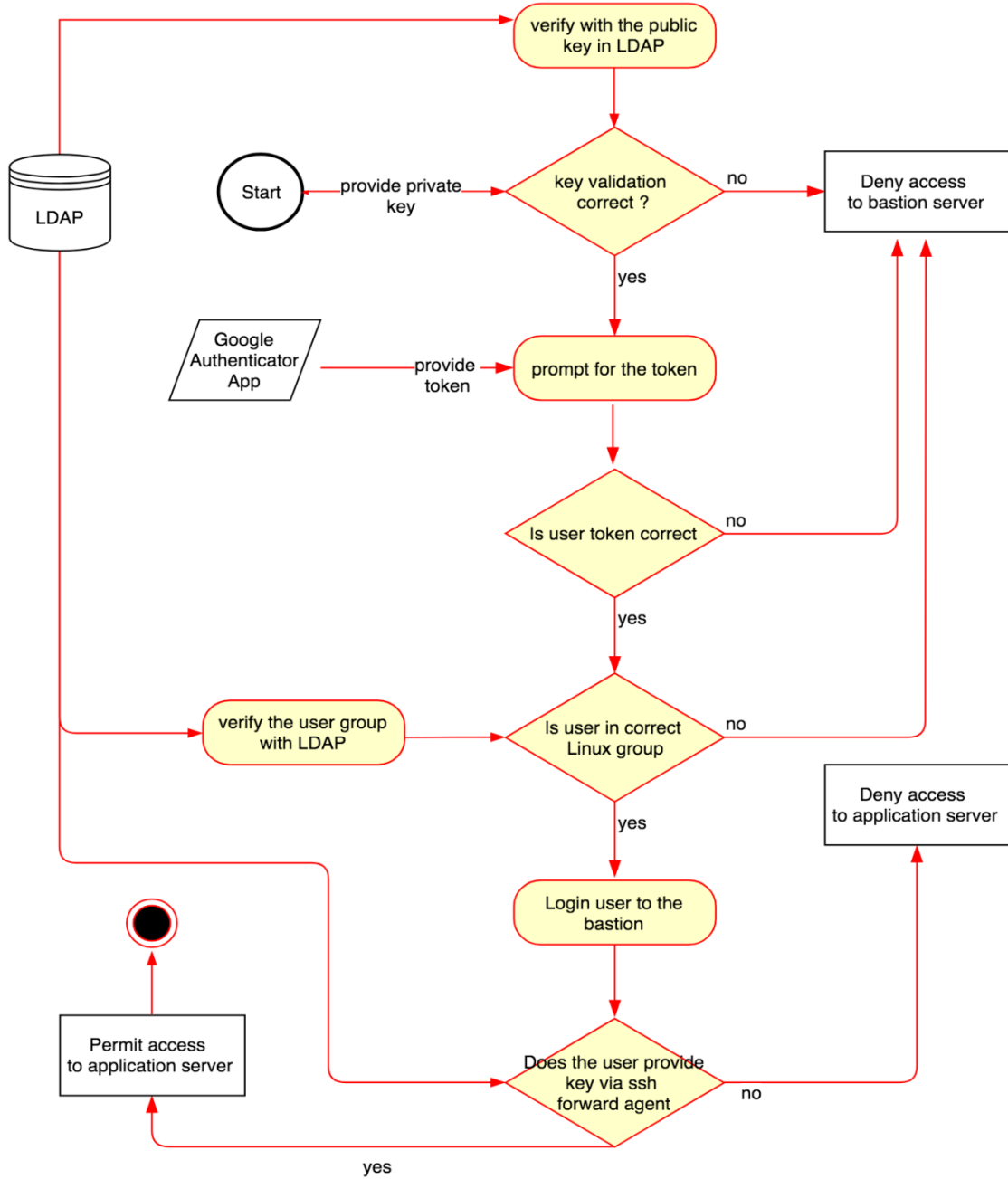


Figure 3.2.5 Flow chart for SSH authentication with MFA

3.3 Design patterns for a Cloud

The core implementation is discussed in previous section. But to implementing this in a larger scaled, enterprise cloud specific environment, it needs to be aligned with a proper design. This section shows the intended pattern used in this implementation and how it is used in a cloud based environment.

3.3.1 Design for a cloud based environment with a Bastion server.

This design pattern (Figure 3.3.1) shows the deployment architecture for a cloud with a bastion host and set of internal private servers. Bastion host is kept in the DMZ or the public subnet and that is the only entry point to the cloud for SSH access. Then anyone who is an authorized user to bastion server will then be able to access the internal servers via SSH. In this deployment LDAP server is deployed in the corporate data center because it needs to be protected against external attacks.

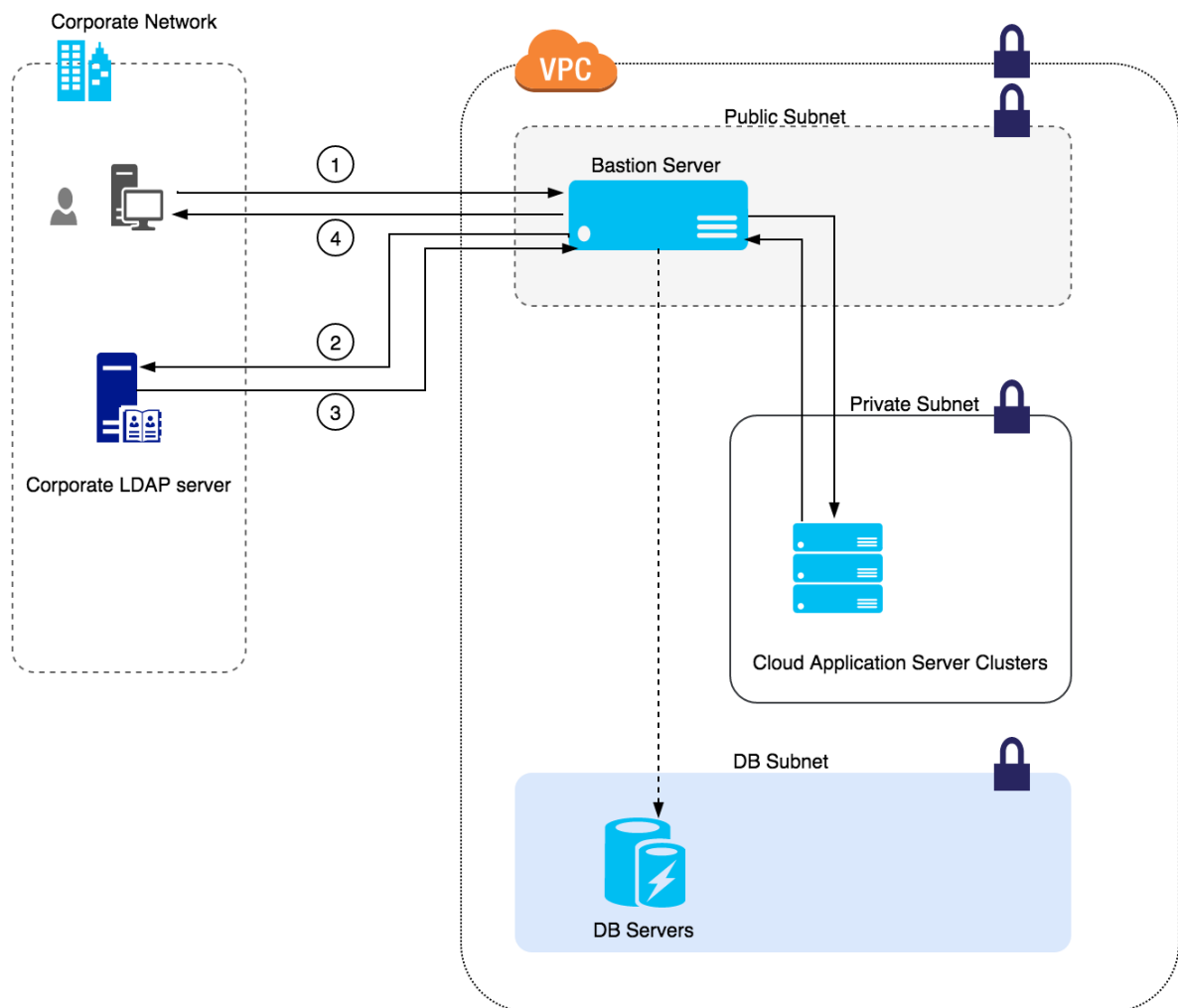


Figure 3.3.1 Standard Cloud setup with a bastion node

3.3.2 Initial Proof of Concept

An initial proof of concept was carried out to check the feasibility of the implementation since it is very important to put effort on a project that is potentially beneficial much upon success. A test environment (depicted in figure 3.3.1) to test the SSH and LDAP integration was setup

using two virtual machines. This design has two Linux Ubuntu 14.04 LTS server version virtual machines running on Oracle Virtual Box. One server hosted the directory service, LDAP and the other was used to emulate the remote server with SSH server. Local machine SSH client was used as the local user device. The two virtual machines are able to communicate within each other via internal network or host only adapter and the host machine can use the same.

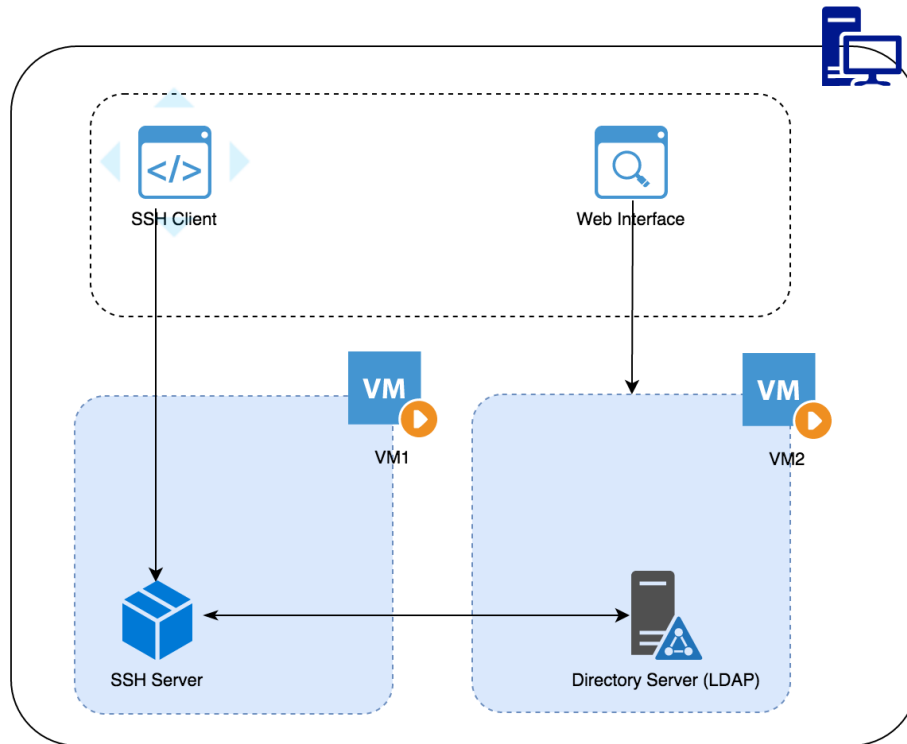


Figure 3.3.1 Initial PoC Setup

Chapter 4: Implementation

The real implementation of the project is discussed in this chapter. That includes how the SSH and LDAP integration is done in the system and the communication implementation, how does the LDAP store the user's public key and how the Google authenticator is used for two step verification. Also the user interface and features are discussed in this chapter.

4.1.1 SSH and LDAP integration

As per the design, the SSH server needs to get the user's public key since it does not store it within authorized key file. To get the users public key from LDAP, SSH should support this functionality and it should have an inbuilt method for that. But the current OpenSSH protocol only supports password based LDAP integration, where it can fetch the user password from the LDAP server. Still fetching public key from another location is not supported as an inbuilt function. Hence we have to use an alternative method. In this context we discuss how we can use currently available features of SSH to query LDAP server and what other additional customizations do we need to do to get the requirement implemented.

4.1.3 Configurations and customizations need to be done in SSH server

The SSH server needs to do LDAP queries and for that ldap-utils package has to be installed, which allows to do LDAP queries using `ldapsearch` command in system level. SSH has set of authentication techniques available inbuilt to that. These techniques can control many behaviors of SSH authentication by setting values or enabling or disabling. In this case SSH `AuthorizedKeyCommand` and `AuthorizedKeyCommandUser` is used to integrate with LDAP and run LDAP queries in system level.

Use of `AuthorizedKeysCommand`

This feature in OpenSSH allows running a defined command or a script during SSH authentication. In this implementation, the advantage of the `AuthorizedKeysCommand` is used to specify a program to be used for lookup of the user's public keys. The program will be invoked with its first argument as the name of the user being authorized, and should produce on standard output `AuthorizedKeys` lines (see `AUTHORIZED_KEYS` in `sshd(8)`). By default (or when set to the empty string) there is no `AuthorizedKeysCommand` to run. If the `AuthorizedKeysCommand` does not successfully authorize the user, authorization falls through to the `AuthorizedKeysFile`. Note that this option has an effect only with `PubkeyAuthentication` method turned on [21].

Use of `AuthorizedKeysCommandRunAs`

Specifies the user under whose account the `AuthorizedKeysCommand` is run. Empty string (the default value) means the user being authorized is used.

4.1.4 Fetching the Public key and the group of the user from LDAP

This is the customization part that needs to be done. We take the advantage of the `AuthorizedKeysCommand` to execute a script which can query the LDAP and retrieve the specific users' public key. The script is given below.

```
#!/bin/bash
cn=$1
server=192.168.57.105 #Put your server IP
basedn=ou=users,dc=ucsc,dc=org #Put your basedn
groupdn=ou=groups,dc=ucsc,dc=org
port=389
google_auth=$(ldapsearch -x -h $server -p $port -o ldif-wrap=no -b
$basedn -s sub "(&(objectClass=posixAccount)(uid=$cn))" | grep
description | cut -d " " -f 2)

[[ -d /home/$1 ]] || mkdir /home/$1
chown -R $1:user /home/$1
chmod 600 /home/$1/.google_authenticator
echo $google_auth | base64 --decode > /home/$1/.google_authenticator
chown -R $1:user /home/$1

userKey=$(ldapsearch -x -h $server -p $port -o ldif-wrap=no -b $basedn -
s sub "(&(objectClass=posixAccount)(uid=$cn))" | sed -n 's/^[
\t]*sshPublicKey:[ \t]*\(.*\)/\1/p')
groupUser=$(ldapsearch -x -h 192.168.57.105 -p 389 -o ldif-wrap=no -b
$groupdn -s sub "(&(objectClass=posixGroup)(gidNumber=5000))" | grep -w
"\<$cn\>" | cut -d " " -f 2)

if [ "$groupUser" = "$cn" ]
    then echo $userKey
    else echo ""
fi
```

The name of the script can be any and can be placed anywhere. In this context the name of the script is `ssh_ldap_query.sh`. That is placed in `/usr/bin/` as `/usr/bin/ssh_ldap_query.sh`

Permission and run user

Make sure this script is only runnable to root user and do not need to be able to access by any other user. Hence providing only read and execute permission should be sufficient to this script to root user. Run below Linux commands to secure the script.

```
chown root:root /usr/bin/ssh_ldap_query.sh
chmod 500 /usr/bin/ssh_ldap_query.sh
```

Then in the `sshd_config` file, need to set the below configurations.

```
AuthorizedKeysCommand /usr/bin/ssh_ldap_query.sh
AuthorizedKeysCommandUser root
```

4.1.5 What does the script do?

This shell script takes the user id (name) of the user trying to log in to the server as an input and do a LDAP query (`ldapsearch`) with the other `basedn` information. The script then checks the user group and match with the proper Linux group. Finally if it is for the first the user logging in, his initial home is created and the configurations related to Google authenticator is stored in the user's home. The data related to these configurations is fetched as a base64 encoded text from the LDAP server. If you execute the search query only, it will return some

other attribute values as well (See Appendix I). To authenticate the user, only Public key of the user is needed. Hence public key can be filtered out of the returned output by piping it to the shell command given below.

```
sed -n 's/^[ \t]*sshPublicKey:[ \t]*\(.*\)/\1/p'
```

This is for the authentication. For authorization, user's group is fetched and it is also compared whether allowed to login to the bastion server. User's group is queried from LDAP server and compared with the corresponding servers group.

4.1.4 Other SSH server configurations

Even the user is authenticated; there are other requirements for him to be correctly logged in to the system. For that the, LDAP utility programs configured in the server has to get the user home, login shell etc. from LDAP server. This can be done by adding configurations to nsswitch.conf.

“The Name Service Switch (NSS) configuration file, /etc/nsswitch.conf, is used by the GNU C Library to determine the sources from which to obtain name-service information in a range of categories, and in what order. Each category of information is identified by a database name. The file is plain ASCII text, with columns separated by spaces or tab characters. The first column specifies the database name. The remaining columns describe the order of sources to query and a limited set of actions that can be performed by lookup result.”[22]

Following configurations need to be done in the mentioned file.

```
# /etc/nsswitch.conf
#
# Example configuration of GNU Name Service Switch functionality.
# If you have the `glibc-doc-reference' and `info' packages installed,
# try:
# `info libc "Name Service Switch"' for information about this file.

passwd:          compat ldap
group:           compat ldap
shadow:         compat ldap

hosts:          files dns
networks:       files

protocols:      db files
services:       db files
ethers:         db files
rpc:            db files

netgroup:       nis
```

Note : The order of **passwd**, **group**, **shadow** is important because of the order is “ldap compat”, then in case if the LDAP server is not available to connect at server startup, the server will go to a hanged state for not being able to get the name-service information.

Next create file “/usr/share/pam-configs/mkhomedir” if not exists and add below configuration.

```
Name: activate mkhomedir
Default: yes
Priority 900
Session-Type: Additional
Session:
    required pam_mkhomedir.so umask=0022/etc/skel
```

This is to setup pam module and the pam_mkhome PAM module create a specific user home folder if it does not exist at login. This is mostly used in scenarios like when LDAP, NIS or Kerberos databases provide user information without using a distributed file system or pre-creating huge number of folders. The /etc/skel helps to copy default files and the umask defines the umask for the user home creation. Also note upon logout, the new user's home directory is not removed. Add below configuration to the end of file /etc/pam.d/common-session.

```
session required pam_mkhome.so skel=/etc/skel umask=0022
```

After doing the given configurations, restart the NSS service by using the command

```
service nscd restart
```

4.1.5 LDAP Server custom configurations

Standard OpenLDAP does not have a public key store capability. Thus we have to import and create new schemas in that LDAP to store public key as an attribute [23].

```
sshPublicKey.schema

attributetype ( 1.3.6.1.4.1.24552.500.1.1.1.13 NAME 'sshPublicKey'
DESC 'MANDATORY: OpenSSH Public key'
EQUALITY octetStringMatch
SYNTAX 1.3.6.1.4.1.1466.115.121.1.40 )
# printableString SYNTAX yes|no
objectclass ( 1.3.6.1.4.1.24552.500.1.1.2.0 NAME 'ldapPublicKey' SUP top
AUXILIARY
DESC 'MANDATORY: OpenSSH LPK objectclass'
MUST ( sshPublicKey $ uid )
)
```

Command to import the schema to the OpenLDAP server.

```
root@LDAPServer:/tmp# ldapadd -Q -Y EXTERNAL -H ldapi:/// -f ssh-
ldap.schema
```

Once this is added to OpenLDAP, you can see there is a new objectClass attribute called “ldapPublicKey” and new attribute called “sshPublicKey”. Now the system administrator can add user’s public key as a new attribute to his LDAP. This is the core implementation and this can be used in cloud based implementation further to provide the required functional deployment.

4.1.6 Google authenticator configurations

The installation of Google Authenticator is described and clearly provided in step by step in digitalocean documentation [16]. Only followed the steps provided there. But integrating it with user interface was a custom code done for this project. This is implemented with the UI by invoking the google-authenticator command via the interface provided with the user detail. This will generate a URL which redirects to Google authenticator API service. This simply provides a QR (Appendix I.3) and that can be scanned using Google Authenticator app in the mobile device to insert the corresponding user data to the mobile device (Figure 4.1.6). Once this is added to the app, it will generate a time based token with a pre-defined interval (Figure 4.1.7). This token can be used to login to the SSH server.

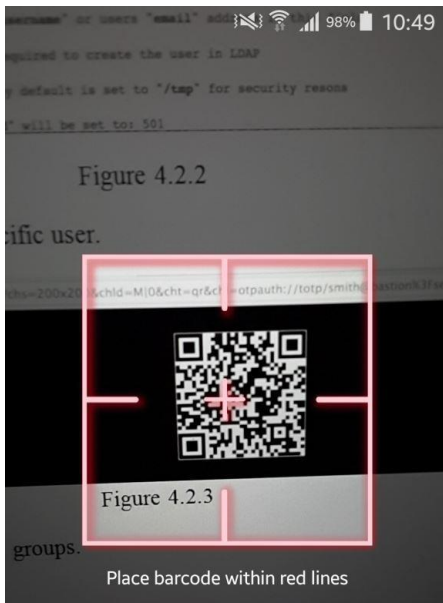


Figure 4.1.6 Scanning the QR code

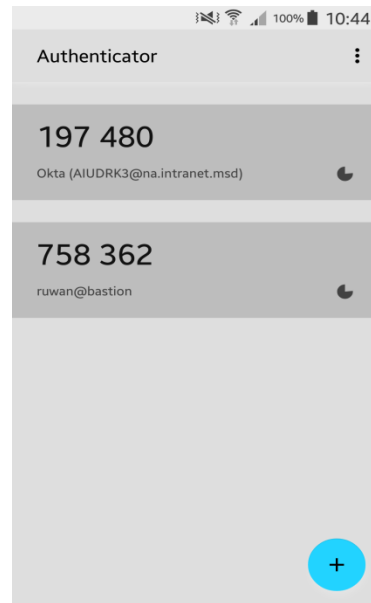


Figure 4.1.7 Time based token generated

Example:- A URL generated from Google authenticator looks like below

```
https://www.google.com/chart?chs=200x200&chld=M|0&cht=qr&chl=otpauth://hotp/
/root@LDAPServer%3Fsecret%3DIOVCHVV3C5H6PP6R
```

This also generates a file that contains matching information which is saved in the machine for a specific user.

```
root@LDAPServer:~# cat /root/.google_authenticator
5B5MMX2Q6ZIBGE2X
" RATE_LIMIT 3 30
" DISALLOW_REUSE
" TOTP_AUTH
81266940
52428740
96276447
41895268
79052761
```

When generating the QR code, a base64 encoded string that contains the matching user information generated in the file `/.google_authenticator` is stored as meta data for the corresponding user. When the user initially logs into a server, the script stored in the remote server (discussed in section 4.1.5) gets these details from the LDAP server and store those configurations in the remote server. Hence, when the user logs into the system, apart from the key validation, a token code is prompted related to his Google Authenticator. It is very important that both the server and the mobile device should be time synced properly to work this because the token is a time based one and expires within a short interval.

4.2 User interface to manage the system operations

4.2.1 User interface

This was one of the defined deliverables of the project and developed using PHP. The User interface has lot of features including creating users, deleting, adding to groups etc. All the details related to the user interface are given below. See Appendix II for user interface screen captures.

4.2.2 Basic functionality of the interface

- Password based authentication to the application itself. (Appendix II : Figure 4.2.1)
- Add new users to the system (Add user metadata through the GUI; name, public key, login-shell, user home etc) (Appendix II : Figure 4.2.2).
- Generate QR code for the specific user (Appendix II : Figure 4.2.3).
- Add/Remove users to/from groups (Appendix II : Figure 4.2.4).
- Delete users from system (Appendix II : Figure 4.2.5).
- View all the users in the system (Appendix II : Figure 4.2.6).
- View users in each group (Appendix II : Figure 4.2.7).

Chapter 5: Results & Evaluation.

5.1 Introduction to the chapter

This is one of the most important phases of this project where the entire effort is tested and to support that, certain test cases were used and results are validated. The tests are taken selectively within the suggested project scope and deliverables. To evaluate the effort put for the implementation, the expected deliverables and end results up to the current stage of the implementation is compared. On the other hand this implementation has to be tested effectively and thoroughly because it is supposed to be implemented as a real-world use case which provides identity management. At the same time this involves essential security characteristics such as authentication and authorization which helps to control different access capabilities to different systems for users. The test plan should have been carried out for both functional and non-functional requirements to assure the best quality of the end result. But for this stage of project and scope, the test cases are limited only for core functionalities.

5.1 Test and validation plan

The test plan contains set of tests for major functional features of the proposed system. These are set of selected tests aligned to the scope and that does not contain nonfunctional test assessments. Each test case is clearly defined and the obtained output is given. The observed output validates the success or fails state of each test case.

5.1.1 Functional Tests

The test plan covers following functional test cases.

1. Testing the SSH public key picking process from LDAP
2. Testing login with a jump box by enabling SSH forward agent.
3. Testing the LDAP user creation process (adding new user to the system).
4. Testing an altered user privilege and login behavior.
5. Carrying out tests for performance evaluation.

5.1.2 Test environment

To execute above mentioned test cases, an improved version of the setup done for the proof of concept was used. The diagram in figure 5.1.2 depicts the test environment specifications.

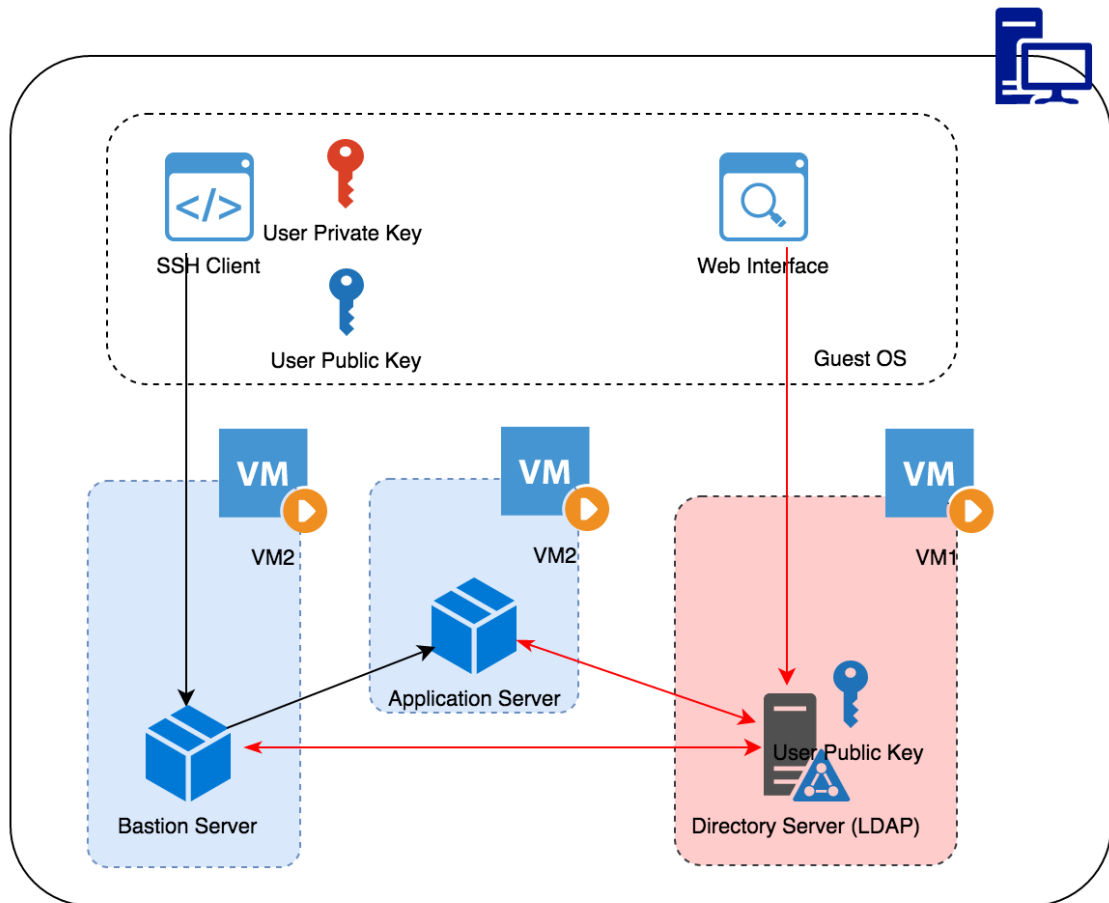


Figure 5.1.2 Local scaled down setup for testing

According to this diagram, local host machine's SSH agent is used as the SSH client and a key pair is created. The three other virtual machines were used as the bastion host, as the application server and as the LDAP server. Specifications of the test environment are given below.

Local Host: Acts as SSH Client

OS version: OS X Version 10.9.5
 SSH Client version: OpenSSH_6.2p2
 SSH key size: 2028 bit

VM-1 : Acts as LDAP Server Host

OS version: Ubuntu 14.04.5 LTS
 SSH Server version: OpenSSH_6.6.1p1
 LDAP version: OpenLDAP-2.4.31

VM-2: Acts as the Bastion Node

OS version: Ubuntu 14.04.5 LTS
 SSH Server version: OpenSSH_6.6.1p1

VM-3: Acts as the Application Server Node

OS version: Ubuntu 14.04.5 LTS
 SSH Server version: OpenSSH_6.6.1p1

5.1.3 Test Execution and results

Test Case 01: Testing the SSH public key picking process from LDAP

This is the basic SSH login and here the test is to see whether the SSH server picks the public key from the LDAP server.

```
User           : anura
SSH Client     : 192.168.57.1
SSH Server    : 192.168.57.102
LDAP Server   : 192.168.57.105
SSH KeyPair   : /tmp/id_rsa_anura and /tmp/id_rsa_anura.pub
```

LDAP Entry for the user “anura” (See Appendix III)

When the user login take place the Linux auth.log was captured and that show the user is successfully authenticated to the system. SSH server Auth Log (/var/log/auth.log) given below.

```
Mar  8 05:53:39 bastion sshd[6253]: Accepted publickey for anura from
192.168.57.1 port 60033 ssh2: RSA
45:3a:e1:32:4c:be:78:90:4e:eb:df:59:e9:03:1e:44
Mar  8 05:53:39 bastion sshd[6253]: pam_unix(sshd:session): session
opened for user anura by (uid=0)
```

SSH Client verbose output (See Appendix IV) also show that the user was able to SSH to the remote server and the authentication was successful. LDAP debug Log (See Appendix V) shows that the SSH server queries for the user data stored for the specific user. This clearly shows that the user was authenticated by using the data stored in the centralized LDAP server. This test results provide that the authentication information which shows user anura’s public key was fetched from LDAP server to authenticate him to the SSH server.

Test case 02: Testing login with a jump box by enabling SSH forward agent

```
User           : anura
SSH Client     : 192.168.57.1
SSH Server (Bastion) : 192.168.57.102
SSH Server (Internal Private Node) : 192.168.57.104
LDAP Server    : 192.168.57.105
```

LDAP Entry for the user “anura” (See Appendix III)

Here first the user adds his private key to the SSH agent using “ssh-add” command. Also in both SSH Client and Bastion we have set “ForwardAgent yes” to forward the key upon login. Below is the SSH login to “bastion” and after that logging into “sshserver” by using bastion as a jump box. See Appendix VI for SSH Client Log.

```
dilan@Dilans-MacBook-Pro:~$ ssh-add /tmp/id_rsa_anura
Identity added: /tmp/id_rsa_anura (/tmp/id_rsa_anura)
```


Bastion auth.Log

```
Mar  8 17:00:28 bastion sshd[9867]: Accepted publickey for anura from
192.168.57.1 port 63591 ssh2: RSA
45:3a:e1:32:4c:be:78:90:4e:eb:df:59:e9:03:1e:44
Mar  8 17:00:28 bastion sshd[9867]: pam_unix(sshd:session): session
opened for user anura by (uid=0)
```

SSHServer Log

```
Mar  8 17:00:54 sshserver sshd[2565]: Accepted publickey for anura from
192.168.57.102 port 44411 ssh2: RSA
45:3a:e1:32:4c:be:78:90:4e:eb:df:59:e9:03:1e:44
Mar  8 17:00:54 sshserver sshd[2565]: pam_unix(sshd:session): session
opened for user anura by (uid=0)
```

This shows that user was able to login to the SSH server from bastion host by forwarding the SSH key.

Test case 03: Testing the LDAP user creation process

A new user called “Charith Kangara” is added to the LDAP server. First an existing users LDAP entry is exported and used as the LDAP template to create the new user. Figure 5.1.3 shows the existing user list.



Figure 5.1.3 LDAP user store

Then we create a new SSH key pair. Figure 5.1.4 (See Appendix VII). After that we export an existing user LDAP template (Figure 5.1.5) and we import the new user with new user information Figure 5.1.6 (See Appendix VIII). Finally we login as the new user. SSH Client debug output given in Appendix IX.1 shows that without Private Key provided, user is not able to login to the system and the next authentication method is prompted by the SSH client. SSH Client debug output given in Appendix IX.2 shows that with Private Key provided, user is successfully able to login to the Bastion server.

Bastion auth.log given below shows the user is authenticated to the server.

```
Mar  8 18:10:15 puppetmaster sshd[10671]: Accepted publickey for charith
from 192.168.57.1 port 64349 ssh2: RSA
6f:5e:c9:a6:8f:47:3f:02:b8:c6:fa:fe:08:df:10:c5
Mar  8 18:10:15 puppetmaster sshd[10671]: pam_unix(sshd:session): session
opened for user charith by (uid=0)
```

Test case 04: Testing an altered user privilege and login behavior

Here we first modify user Charith's "OU" and check the access. In the SSH server we allow only the "users" OU and here we change user Charith's OU . Figure 5.1.9 shows the LDAP alter LDIF file and Figure 5.1.10 shows the output upon running this modify query to LDAP server.

```
# LDIF Export for cn=charith kangara,ou=users,dc=ucsc,dc=org
# Server: My LDAP Server (192.168.57.105)
# Search Scope: base
# Search Filter: (objectClass=*)
# Total Entries: 1
#
# Generated by phpLDAPadmin (http://phpldapadmin.sourceforge.net) on March 8, 2017 5:04 am
# Version: 1.2.2

version: 1
# Entry 1: cn=charith kangara,ou=users,dc=ucsc,dc=org
dn: cn=charith kangara,ou=users,dc=ucsc,dc=org
changetype: moddn
newrdn: cn=charith kangara
deleteoldrdn: 0
newsuperior: ou=mis,dc=ucsc,dc=org
```

Figure 5.1.9 LDAP modify user OU in LDIF

```
root@LDAPServer:~# ldapmodify -h 192.168.57.105 -D "cn=admin,dc=ucsc,dc=org" -W -f charith.ldif
Enter LDAP Password:
modifying rdn of entry "cn=charith kangara,ou=users,dc=ucsc,dc=org"
```

Figure 5.1.10 LDAP modify user OU command line

Then we can see the user's "OU" is changed (Figure 5.1.11) And when user tries to login, authentication fails (Figure 5.1.12) because the user is not in the intended LDAP OU

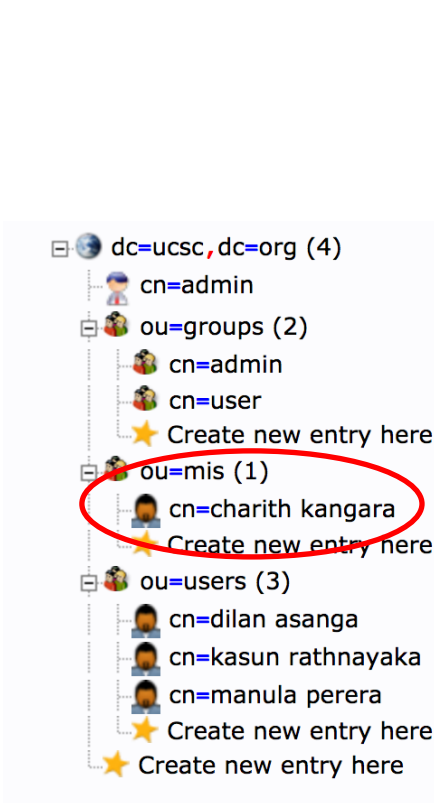


Figure 5.1.11 Change user's OU

```
dilan@Dilans-MacBook-Pro:~$ ssh -v charith@192.168.57.102 -i /tmp/id_rsa_charith
OpenSSH_6.2p2, OpenSSL 0.9.8r 8 Dec 2011
debug1: Reading configuration data /Users/dilan/.ssh/config
debug1: Reading configuration data /etc/ssh_config
debug1: /etc/ssh_config line 20: Applying options for *
debug1: /etc/ssh_config line 102: Applying options for *
debug1: Connecting to 192.168.57.102 [192.168.57.102] port 22.
debug1: Connection established.
debug1: identity file /tmp/id_rsa_charith type 1
debug1: identity file /tmp/id_rsa_charith-cert type -1
debug1: Enabling compatibility mode for protocol 2.0
debug1: Local version string SSH-2.0-OpenSSH_6.2
debug1: Remote protocol version 2.0, remote software version OpenSSH_6.6.1p1 Ubuntu
debug1: match: OpenSSH_6.6.1p1 Ubuntu-Zubuntu2.8 pat OpenSSH*
debug1: SSH2_MSG_KEXINIT sent
debug1: SSH2_MSG_KEXINIT received
debug1: kex: server->client aes128-ctr hmac-md5-etm@openssh.com none
debug1: kex: client->server aes128-ctr hmac-md5-etm@openssh.com none
debug1: SSH2_MSG_KEX_DH_GEX_REQUEST(1024<1024<8192) sent
debug1: expecting SSH2_MSG_KEX_DH_GEX_GROUP
debug1: SSH2_MSG_KEX_DH_GEX_INIT sent
debug1: expecting SSH2_MSG_KEX_DH_GEX_REPLY
debug1: Server host key: RSA a1:53:b1:71:f4:dc:50:51:ed:36:6e:01:ea:93:c3:8c
debug1: Host '192.168.57.102' is known and matches the RSA host key.
debug1: Found key in /Users/dilan/.ssh/known_hosts:72
debug1: ssh_rsa_verify: signature correct
debug1: SSH2_MSG_NEWKEYS sent
debug1: expecting SSH2_MSG_NEWKEYS
debug1: SSH2_MSG_NEWKEYS received
debug1: Roaming not allowed by server
debug1: SSH2_MSG_SERVICE_REQUEST sent
debug1: SSH2_MSG_SERVICE_ACCEPT received
debug1: Authentications that can continue: publickey,password
debug1: Next authentication method: publickey
debug1: Offering RSA public key: /tmp/id_rsa_anura
debug1: Authentications that can continue: publickey,password
debug1: Offering RSA public key: /tmp/id_rsa_charith
debug1: Authentications that can continue: publickey,password
debug1: Next authentication method: password
charith@192.168.57.102's password: |
```

Figure 5.1.12 Authentication Failure

Test case 05: Testing network level capabilities and limitations.

Here we test how system behaves when network connectivity between SSH server and LDAP server is not working properly. If the LDAP service is down, then the SSH server is unable to get the public key. Hence the login fails (See Appendix X for the output). We can test this just by shutting down the LDAP server.

5.2 Test Summary and Validation

These test cases address the main deliverables of this project. The first test case validate the core implementation which is to check whether we can pick the public key stored in an LDAP server upon login without storing it in the server itself. The test result was successful and when a new user tries to access the server with his private key, the server got the corresponding username and queried the LDAP server for the corresponding public key and accepted the user upon validation.

Second test case checks the cloud based requirement, which is to use a bastion as a jump box to the instances in internal private subnet. For both login sessions we used the LDAP server and the public key provider. The agent had to enable SSH forward agent to do the multiple instance authentication with the same credentials. In this test case user was able to log into the bastion node first and then from the bastion node without specifically providing any key, he was able to login to the application server as well with SSH forward agent enabled on both SSH client machine and bastion.

Third test case was to simply create a new user in LDAP with a public key stored as an attribute. We simply added the user by providing an LDIF input to the LDAP server. Then the SSH client who is having the correct private key is able to SSH to the remote server. We did not have to do anything in remote server other than it is pre-configured just to read the LDAP server. This test indirectly proves and supports the auto scale concept where server doesn't have to store any user specific configuration on the server at any time and upon first login if the user does not already exist, the user home is created and access is provisioned.

Test case four checks where how we can limit the access to a specific user just by changing attributes in LDAP level. This proved that, just by changing the OU of the user, we are able to control access to any user to the system.

Test case 5 shows if the SSH server is unable to communicate with the LDAP server, the authentication fails since the SSH server cannot fetch the Public Key from LDAP. Hence any outage of LDAP would not create any security issue.

5.3 Evaluation

This project is focused on a cloud based environment where lot of users need to connect to large number of remote servers in a managed infrastructure and the servers are scaled up and down timely. Hence the evaluation is done considering these factors.

5.3.1 Evaluation criteria

- Security
- Manageability
- Performance

5.3.2 Evaluated systems

First 3 systems given below are reference systems to the fourth one, which is the main implementation of this project.

- SSH Password based authentication
- SSH Key Based authentication
- LDAP based centrally managed password based authentication
- LDAP based centrally managed key based authentication

5.3.3 Evaluation of each system based on the selected criteria

- **SSH Password based authentication**

Security

Since a password is used as the authentication mechanism, it is always vulnerable to both brute force and dictionary attack [24,25]. The risk can be reduced by using strong passwords with different characters, number and symbol combinations. But it is still vulnerable to key loggers, phishing and shoulder surfing attacks. Remembering a strong password is extremely difficult and since that people use to write down somewhere or store within the system as plain text. This can then be vulnerable to theft of device or file system attack.

Manageability

Managing passwords for large number of servers is difficult (in SSH context). User has to remember or write down password for each server or use a common password. Using different passwords for different servers is secure but it is difficult to manage. Using a single password is efficient, but not secure. Also if user forgets the password, system administrators have to change the password in each and every server which makes it quite unmanageable. In a cloud based environment existing servers may get killed while new servers may spin up. Each time a new server starts, access to all the relevant users has to be granted either manually or using automated tools.

Performance

In SSH context, passwords are normally typed into the terminal or sometimes stored in a terminal program. Comparatively the payload of a password is always less than a key. Hence authentication process could be faster. But with the real world hardware and networks are more than enough to carry out the authentication in less than milliseconds. But every time the user logs into the system, if the password has to be typed then that it quite inefficient. Also even password is stored in a local program when it comes to the bastion concept, logging from bastion to application server doesn't support the stored passwords. So this includes several keyboard interactions when logging into remote servers. Thus it is inefficient.

- **SSH Key Based authentication**

Security

Key based authentication is more secure than passwords (discussed earlier in chapter 2.2.1) [26]. When password access is disabled, most of the automated attacks can be avoided. Still keys are vulnerable to theft of device or compromised file system. Best way is to avoid using shared keys and use per user key. Also keys can be protected using a passphrase. Also keys are resistant to key loggers and

Manageability

In the context of SSH and cloud based environment even managing per user keys for large number of servers for a scaled environment is difficult. Each time a new user is added or removed, his public key has to be added or removed from the `authorize_key` file in the remote server. This makes it difficult to manage just like passwords.

Performance

Once the private and public keys are configured user do not have to provide the key again and again. With `ssh_forward` agent enabled, users are able to login to the application servers from bastion even without copying the key to the bastion server. This reduces the keyboard interaction compared to password. Key validation is slower than password validation. But this is negligible and takes less than a second within the same network and few seconds for remote cloud environments (see test results).

- **LDAP based centrally managed password based authentication**

Security

All the facts given for SSH password based authentication is still valid for this context from user side. Also in this case LDAP server by default stores user passwords and as per the RFC4519 [27,28] specification these passwords are not in encrypted form or hashed format.

Also the `userPassword` attribute is allowed to have more than one entry with different hash functions. It is highly recommended to store passwords in hashed format and even hashed, to be protected as if they were stored in clear text. Because even hashed passwords can be exploited using dictionary and brute force attacks. Still the advantage of a hashed password is the attacker does not have direct access.

Manageability

Since user and credentials are centrally stored in LDAP server, the manageability increases and it is a huge advantage for system administrators to manage users. Also user meta data (home, login shell, uid etc) is stored in LDAP server and that makes managing users in large scaled server infrastructures and cloud based systems much easier. In an auto scaled, cloud based environment, user and access management can be done with this kind of centralized user management system.

Performance

User authentication is validated against LDAP based credentials and this could be an additional hop. Every time a user logs into the system authentication information has to be validated with the authentication details available in the LDAP server. But LDAP is a read optimized system and hence this is negligible and takes less than a second within the same network and few seconds for remote cloud environments (see test results2).

- **LDAP based centrally managed key based authentication**

Security

This is the core of this project and instead of password based authentication, key based authentication is used. The advantage here is user's public key is stored in the LDAP server and that is not a huge security risk unless it is modified.

Manageability

Manageability is same as mentioned in "LDAP based centrally managed password based authentication".

Performance

Performance is same as mentioned in "LDAP based centrally managed password based authentication". Test results are provided.

5.4 SSH login performance evaluation

In this context a critical analysis on SSH login performance for different authentication mechanisms is carried out to evaluate the efficiency of the proposed system (LDAP public key based SSH login). Three reference systems are used to compare the proposed system and they are "SSH Password based authentication", "SSH Key Based authentication" and "LDAP based centrally managed password based authentication". For this, an automated SSH login and logout script is used with different options and tested for time taken per each session (measured in milliseconds). Each script authenticates to the system with provided authentication method and exists itself. This process is repeated and the time taken for each login was gathered. This test was run in 3 different ways to emulate 3 different real world scenarios. They are as follow.

1. A single user tries to login to Bastion server with given 4 different evaluation methods.
2. Multiple users connected to Bastion on different terminal sessions try to login to another server with 4 different evaluation methods.
3. Multiple users from multiple devices trying to login to another server with 4 different evaluation methods.

Below are the 4 scripts that were used for the performance tests.

- **SSH Password based authentication**

This test script uses a password to authenticate to the system. To automatically pass the password to the login prompt, a simple Linux tool named “sshpass” [29] was used so that there is no need to manually type the password when the test is running.

```
for i in {1..30}
do
    START_TIME=`echo $(( $(date +%s%N)/1000000 ))`
    sshpass -p "dilan" ssh -o StrictHostKeyChecking=no
dilan@192.168.57.102 exit 2>&1
    END_TIME=`echo $(( $(date +%s%N)/1000000 ))`
    ELAPSED_TIME=$(( $END_TIME - $START_TIME ))
    echo $ELAPSED_TIME
done
```

- **SSH Key Based authentication**

RSA key pair was created and public key was added to the remote servers authorized_key file. This allows user to login without password by using his private key. This is key based authentication.

```
for i in {1..30}
do
    START_TIME=`echo $(( $(date +%s%N)/1000000 ))`
    ssh -i id_rsa dilan@192.168.57.102 exit 2>&1
    END_TIME=`echo $(( $(date +%s%N)/1000000 ))`
    ELAPSED_TIME=$(( $END_TIME - $START_TIME ))
    echo $ELAPSED_TIME
done
```

- **LDAP based centrally managed password based authentication**

A new user with password stored in the LDAP server was created and this evaluates the centrally managed password based authentication.

```
for i in {1..30}
do
    START_TIME=`echo $(( $(date +%s%N)/1000000 ))`
    sshpass -p "password" ssh -o StrictHostKeyChecking=no
test_password@192.168.57.102 exit 2>&1
    END_TIME=`echo $(( $(date +%s%N)/1000000 ))`
    ELAPSED_TIME=$(( $END_TIME - $START_TIME ))
    echo $ELAPSED_TIME
done
```

- **LDAP based centrally managed key based authentication**

This is the script used to evaluate the proposed system. A test user is created in LDAP and the local user's public key is stored in the LDAP server.

```
for i in {1..30}
do
START_TIME=`echo $(($(date +%s%N)/1000000))`
ssh -i id_rsa test_key@192.168.57.102 exit 2>&1
END_TIME=`echo $(($(date +%s%N)/1000000))`
ELAPSED_TIME=$((END_TIME - START_TIME))
echo $ELAPSED_TIME
done
```

5.4.1 Test scenario 1

A single user tries to login from one machine with 4 different evaluation methods.

Both Client and server machines are based on following specifications

- Platform Infrastructure : Oracle VirtualBox VM
- OS : Ubuntu Linux 14.04
- CPU : 1 vCPU
- Memory : 1024Mb
- Client IP : 192.168.57.171
- Server IP : 192.168.57.102

Each automated script perform 30 SSH login attempts and the corresponding time taken was recorded. Then the time to login was graphed against the number of attempts. All 4 evaluation methods were tried this way and below are the graph for each evaluation method.

SSH with password, key, LDAP password, LDAP key

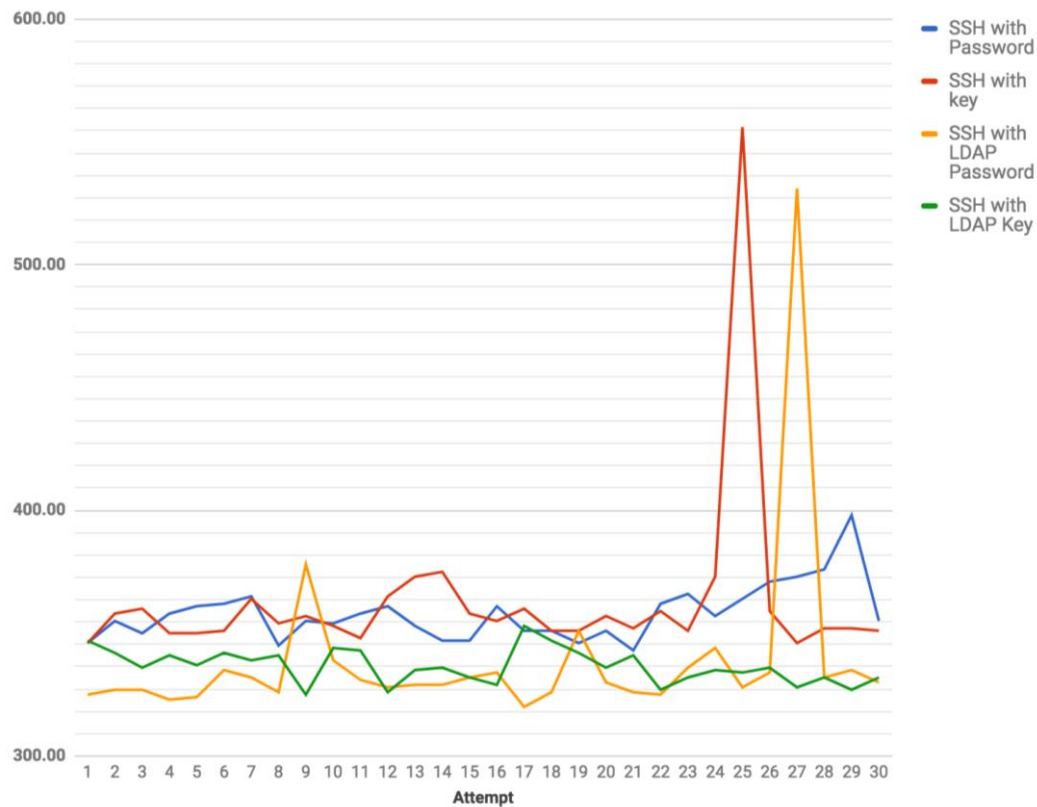


Figure 5.4.1- SSH with password, key, LDAP Password, LDAP key

5.4.2 Test scenario 2

Multiple users connected to bastion try to login to another server with 4 different evaluation methods.

Both Client and server machines are based on following specifications

- Platform Infrastructure : Oracle VirtualBox VM
- OS : Ubuntu Linux 14.04
- CPU : 1 vCPU
- Memory : 1024Mb
- Client IP : 192.168.57.171
- Server IP : 192.168.57.102
- 4 Terminal sessions from the SSH client to test 4 evaluation methods.

- 1) SSH to another server with password. Time taken for each login attempt on 4 terminal sessions of a single SSH client machine is graphed below

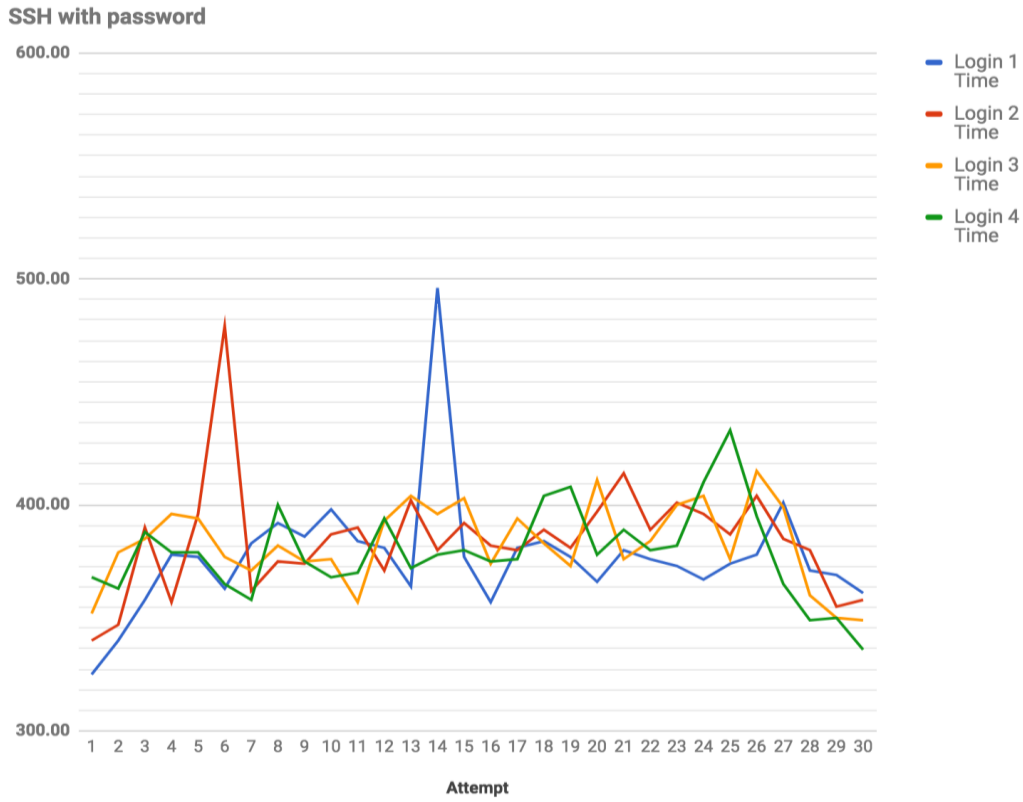


Figure 5.4.2.1 – SSH with password

2) SSH to another server with key. Time taken for each login attempt on 4 terminal sessions of a single SSH client machine is graphed below

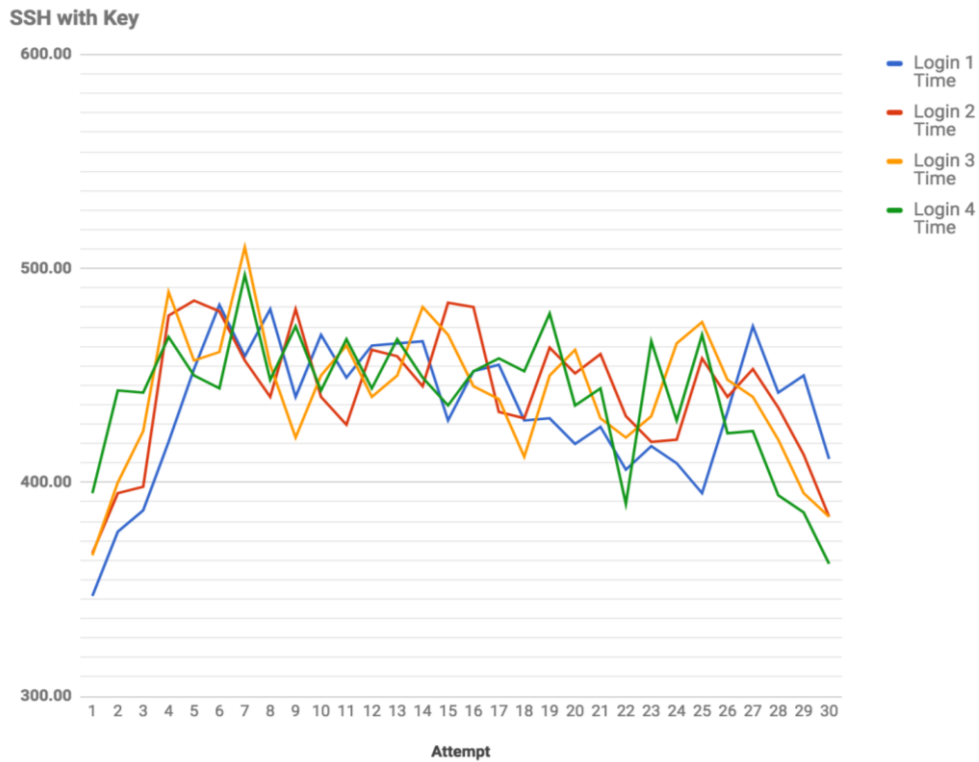


Figure 5.4.2.2 – SSH with key

3) SSH to another server with password stored in LDAP. Time taken for each login attempt on 4 terminal sessions of a single SSH client machine is graphed below

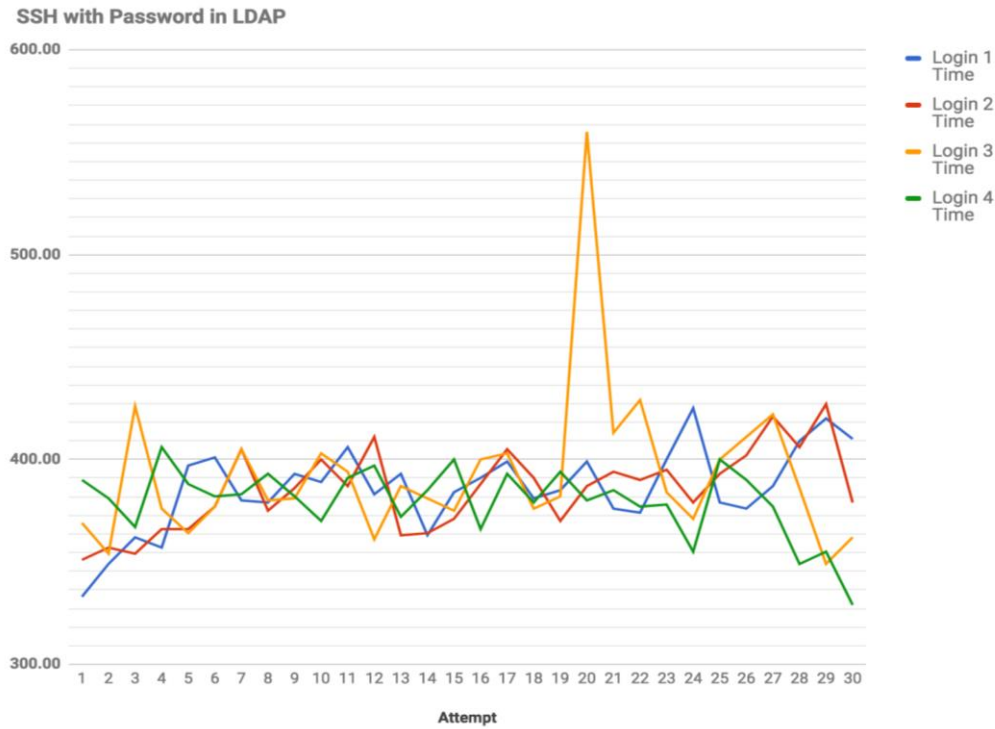


Figure 5.4.2.3

4) SSH to another server with public key stored in LDAP. Time taken for each login attempt on 4 terminal sessions of a single SSH client machine is graphed below.

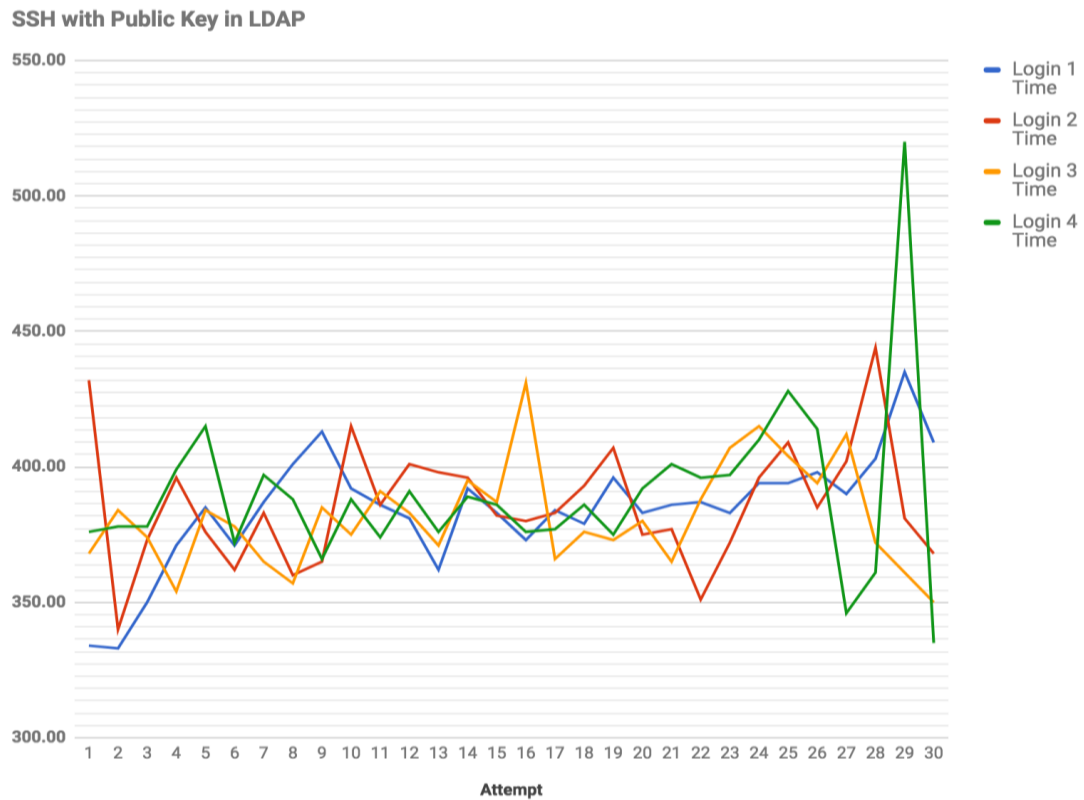


Figure 5.4.2.4

5.4.3 Test scenario 3

Multiple users from multiple devices trying to login to another server with 4 different evaluation methods.

1 server and 4 client machine are based on following specifications

- Platform Infrastructure : Oracle VirtualBox VM
- OS : Ubuntu Linux 14.04
- CPU : 1 vCPU
- Memory : 1024Mb
- Client1 IP : 192.168.57.170
- Client2 IP : 192.168.57.172
- Client3 IP : 192.168.57.104
- Client4 IP : 192.168.57.171
- Server IP : 192.168.57.102
- 1 Terminal sessions from each SSH client to test 4 evaluation methods.

1. SSH to another server with password. Time taken for each login attempt on 4 terminal sessions of 4 different SSH client machines are graphed below

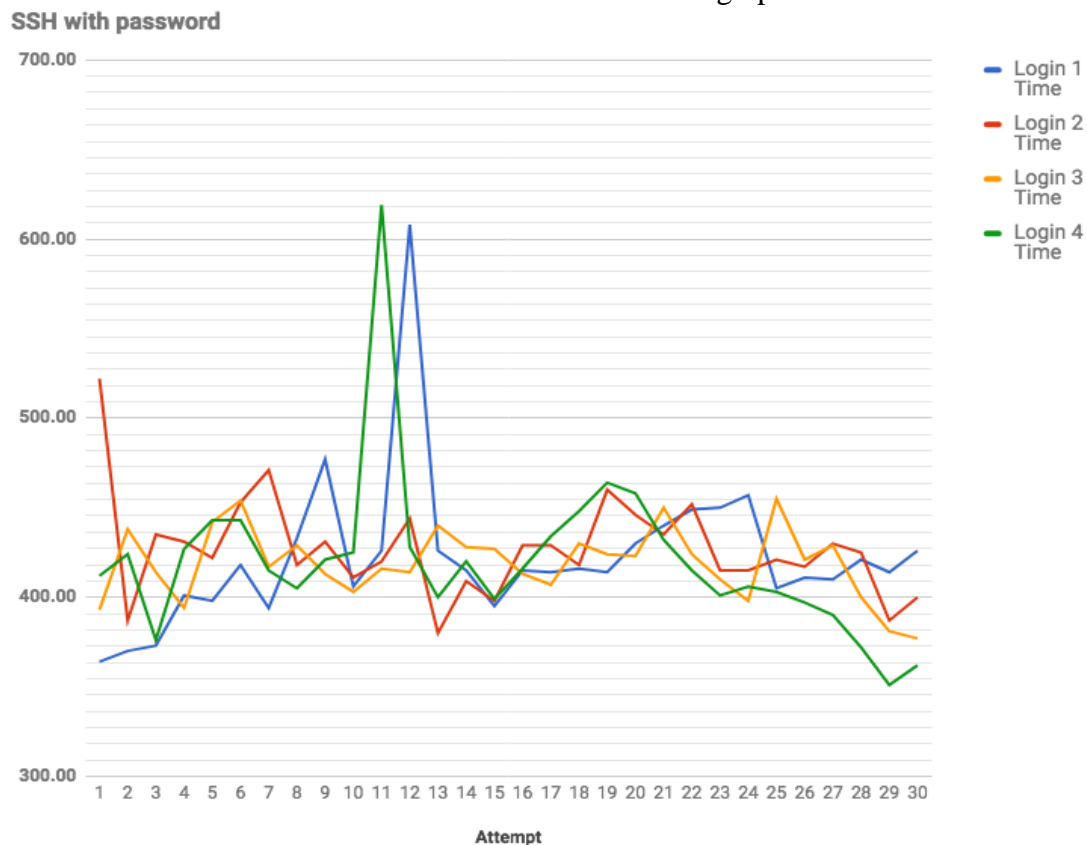


Figure 5.4.3.1

2. SSH to another server with key. Time taken for each login attempt on 4 terminal sessions of 4 different SSH client machines are graphed below

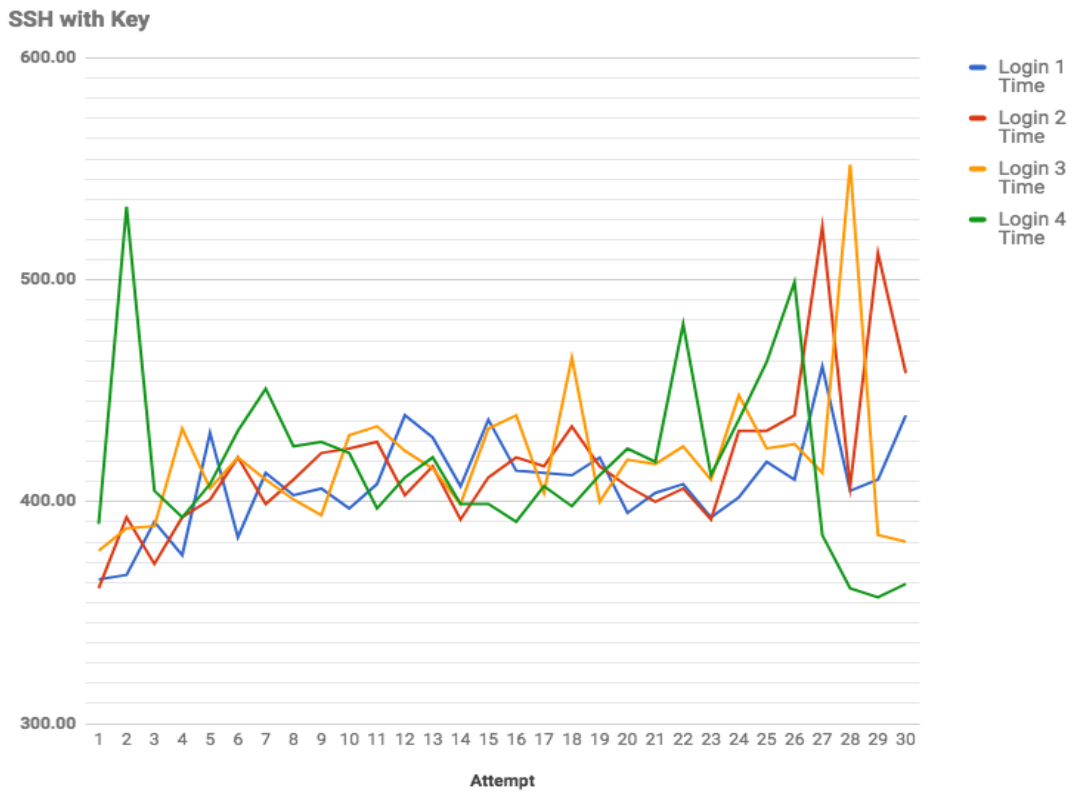


Figure 5.4.3.2

- SSH to another server with password stored in LDAP server. Time taken for each login attempt on 4 terminal sessions of 4 different SSH client machines are graphed below

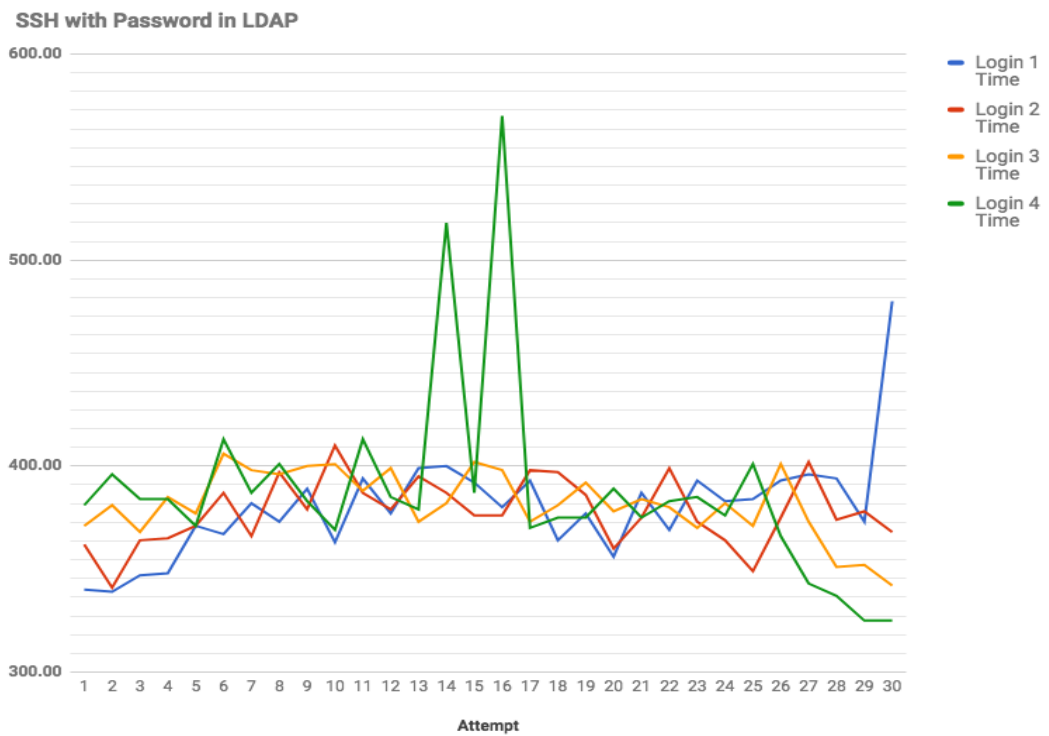


Figure 5.4.3.3

- 4. SSH to another server with public key stored in LDAP server. Time taken for each login attempt on 4 terminal sessions of 4 different SSH client machines are graphed below

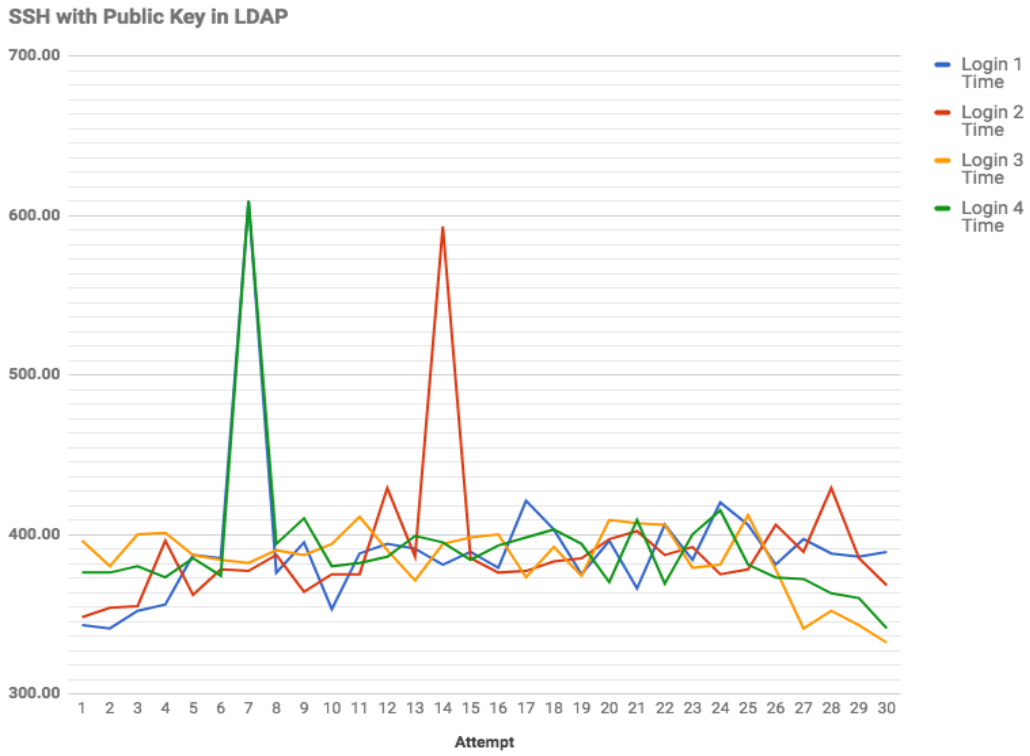


Figure 5.4.3.4

5.5 Performance evaluation summary

Based on the stats collected on 3 different use case scenarios for LDAP based public key SSH authentication, it shows that there is no visible performance degradation or improvement. But based on the graphs it slightly show performance improvement in validating credentials against LDAP stored passwords or keys rather than the locally stored credentials. Hence considering the comparison of centrally managed key based authentication with other 3 reference systems, we can come to a conclusion that it is performing well with the LDAP just as other authentication methods.

The project had couple of goals which is discussed in chapter 1 under sub topic 1.5. Here we evaluate how successfully they are achieved by comparing the outputs of the test results. The projects' main approach was to design a centralized identity and user management mechanism for public key based SSH authentication and that was aimed for a cloud based environment. The major challenge was to store user identities in a centralized place and using them without being need to store at the server level and provisioning on demand access for both new and already existing users to the system. Also the aim was to manage them from a single location. These goals have delivered and that was validated in test case one, three and four. So the main goal is achieved and the test cases were successfully validated using a scaled down version of the real implementation.

The other goal was to design and implement this in a cloud-ready manner and align the project with basic requirements of a cloud based environment. In the design specification we discussed how good the proposed system can cater access and identity management for a cloud environment with SSH. For that we initially analyzed what major technologies used a cloud environment to access resources. There we found that especially in Linux based environments, it is highly recommended to use a bastion instead of directly exposing the application servers to the outside. The next challenge was to design and implement the suggested solution with the bastion concept. There we had to assure a user can still use the same central identity to connect both bastion and then from there to the internal private server seamlessly without having to do changes or do operational level things. The suggested design is capable of authenticating user to both bastion and internal server using the same central identity management mechanism and that was tested and evaluated in test case two and three.

We also faced another challenge specific to a cloud based environment, which is auto scaling. The challenge was user cannot expect a given server to stay permanently. Also in a high available type cloud where the system support threshold based auto scaling, new servers will be started and old might get terminated. If the systems engineer has to configure user public keys every time a new server starts and creates users, which is not going to be scalable and efficient. Also when a new user comes to the system, he or she needs to be given access to each server. But in this proposed system, users and identities are need to be managed in LDAP only and users who are in correct LDAP OU are allowed to SSH to the environment. This is a valuable goal that was achieved and is much applicable and useful in a cloud based environment. We tested this successfully in test case two, three and four. Specially, test case four proved that a given users access can be controlled centrally at LDAP level and that is an important goal that was achieved in this project.

Chapter 6: Conclusion and Future Work

6.1 Problems Faced

At the initial stages of this implementation project, had to go through existing systems and had to learn how they work. Also finalizing the project deliverables was difficult at initial stage since this is an implementation project. Compared with a research project, an implementation project is supposed to deliver a working system or functionalities at the end, which was a challenging task.

The next problem was to find research papers and articles focused on the specific implementation. Even though there were several documentations based on individual core technologies, there were less related ones to the integrated solution. So the author had to try out different approaches during the initial proof of concept.

Default OpenLDAP installation did not have an attribute to store a public key. Different schemas were found when searching in internet, but many of them did not work out of the box as suggested in the documentation. After multiple tryouts was able to find the correct schema.

Even though managed to query and get the users public key upon login attempt, it did not work for some reason at the beginning. Later found that the key was not in correct string format and need to use `ldif-wrap=no` switch with `ldapsearch` command.

Some other difficulties were found when integrating the Google Authenticator QR code generation with the user interface developed for the user management. Later found that this was due to trying to include HTTPS content within HTTP contents. As a fix, the entire site was converted to HTTPS based web interface and had to use JavaScript and jQuery for the user interface development.

6.2 Deviation from original project plan

In the original project proposal that included a suggestion to evaluate secure LDAP to communicate between LDAP replicas, but was not able to evaluate at this phase of development work. Also WSO2 Identity server was suggested to evaluate in this project for identity management. But that was not able to evaluate at this phase. Finally the initial proposal had a phase to implement a user activity and tracking solution. But that is quite different from the main scope and was not added as a part of the implementation.

6.3 Future work

What other features and capabilities that can be implemented, integrated is discussed here.

- Implement admin, standard and read-only user management.
This is related to Linux level access control, but need to evaluate how feasible to map LDAP groups or OUs can be mapped with Linux user groups so that different level of access can be provided.
- Implement LDAPS (secure LDAP). The cloud based deployment can be secured using the secure LDAP communication between the cloud LDAP server and the corporate LDAP server.

6.4 Conclusion

This implementation project has delivered its suggested core functionalities and goals successfully and they are implemented and tested successfully using a scaled down version of the real-world use case. It was very interesting to identify an existing requirement that is common to most of infrastructure which needs proper identity management and access control centrally, and attempt to implement a solution that makes systems administrator's life much easier. Not only that this concept can be easily integrated to any standard IT infrastructures which simply need an identity management solution to manage large number of servers, because the suggested implementation use set of known common technologies to mostly used in many systems. This implementation was intentionally planed for a cloud based environment since cloud technologies are evolving and used widely these days. It was very effective to implement the proposed solution for a cloud based environment since the author was familiar with cloud based environments and has hands on experience in cloud technologies. Thus when implementing this solution author critically analyze the cloud architectures and address the issue related to cloud based environment. Being focus on cloud does not mean that this is not recommended for an in-house server infrastructure. The concepts are the same and only the deployment patterns and network level changes there in both cloud and in house deployment. This project is not just a prototype; it can be used for a real world deployment and the author used a real world use case (the Cloud deployment with large number of servers within his current work place) to collect requirement and address the real problem with a practical approach.

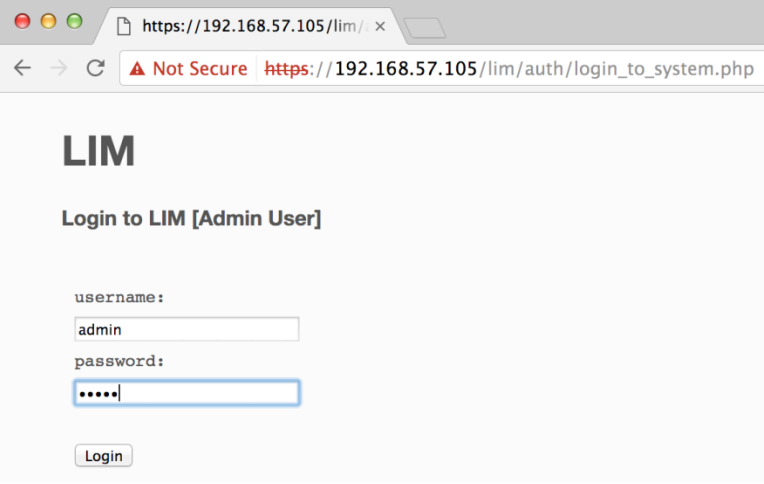
Before starting the implementation work, a good review on available literature and readings were done to identify the most feasible and practical approach. Before selecting a certain authentication methods, a thorough compare and contrast was done to ensure the security. The core implementation items of the proposed scope was covered including the key based SSH authentication with LDAP stored public key, user interface to manage users and integrate with the Bastion concept. Not only that as an additional security layer, Google Authenticator is integrated so that the proposed system is safe even with theft of device.

The testing and evaluation done for the project with statistical data provides very good analytical information and it has clearly shown that the integration has not made any performance degradation, instead a slight performance growth can be seen. That is very good in this kind of system and even with custom implementations, achieving such performance based results is tremendous.

Appendix

Appendix I: User Management Interface

1). Password based authentication to the application itself.



The screenshot shows a web browser window with the following details:

- Address bar: `https://192.168.57.105/lim/`
- Page title: **LIM**
- Page content: **Login to LIM [Admin User]**
- Form fields:
 - username:
 - password:
- Buttons:

Figure 4.2.1

2). Add new users to the system (Add user metadata through the GUI; name, public key, login-shell, user home etc)

username / common name (cn):

General

Home

Groups

List All Groups

Create Group

Delete Group

Add Users To Group

List Users In Group

Delete Users From Group

Users

Search Users

Create User

Edit User

Delete User

Import User

Servers

Add LDAP Server

Modify LDAP Server

Delete LDAP Server

Add/Edit Base Server

username / common name (cn):

surname (sn):

homeDirectory:

loginshell:

email:

password:

ssh public key:

```
ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAQDFmNfcej6H881iWXRaKf9qqp5PxUeKGUcQ/ldE7bYIQz96+mxOGyED9Yz3D93/SEGZU2Qy
nu8ssU+mhts/y7NVL2tVR1dBiQtabwbCqtQL7JpgWqAlfCcMjNbyQVyzxWj53GEh7Hy14cS7eF2KT1+Dahrl15v+wm+cUGK0K8Pvav0ivO
9ePBbG+Md3n6lDoN2opWm4QJl7aaqN7dr3lwnVXODtLY/wOzvpCDlozN6dNuYy/zzTVB+FDtURbG4YJiOsR89HvbVRouEO2pRcENggW
4symZiwtsl6c0kvDoc/XIOWPp6iW4IOrBJPS+FOtQBjed6e9Ut8/rjekxZ smith@gmail.com
```

Google Auth Code (B-64 encoded):

```
QkU2VvkZjVzdUV1oyUjI3NAoifJBVEVFTEINSVQgMyAzMAoiERJU0FMTE9XX1JFVNFNCiigVE9UUF9BVRVICjYwMjUzZmZwCjQ2OTUxMTEz
CjY1MTcwMTE5CjQ5NDU5NjAwCjM4MTc5OTg2Cg==
```

NOTE

- 1) common name "cn" will be used as "username" on client servers, you can use either "username" or users "email" address in this field
- 2) surname "sn" is required to create the user in LDAP
- 3) "homeDirectory" by default is set to "/tmp" for security reasons
- 4) Default "group id" will be set to: 501

Figure 4.2.2

3). Generate QR code for the specific user.



Figure 4.2.3

4). Add/Remove users to/from groups.

← → ↻ **Not Secure** <https://192.168.57.105/lim/groups/manage-group-users-add.php>

LIM

Add Users To Group

Username: admin BaseDN: ou=groups,dc=ucsc,dc=org Server: 192.168.57.105

General Select a group to add users

Home Select a group

Groups admin ▾

Submit

[List All Groups](#)

[Create Group](#)

[Delete Group](#)

[Add Users To Group](#)

[List Users In Group](#)

[Delete Users From Group](#)

Figure 4.2.4

5). Delete users from system.

← → ↻ **Not Secure** <https://192.168.57.105/lim/groups/delusers-from-group.php>

LIM

Delete Users From Group

192.168.57.105 says:
Are you sure you want to delete manula user?

Cancel OK

Username: admin BaseDN: ou=groups,dc=ucsc,dc=org Server: 192.168.57.105

General Click on a "Username" in the table to delete

Home

Groups

[List All Groups](#)

[Create Group](#)

[Delete Group](#)

[Add Users To Group](#)

[List Users In Group](#)

[Delete Users From Group](#)

Group Members
manula
...
...
...
...
...

Figure 4.2.5

6). View all the users in the system.

← → ↻ ▲ Not Secure <https://192.168.57.105/lim/users/editusers.php>

LIM

Edit User

Username: admin **BaseDN:** ou=users,dc=ucsc,dc=org **Server:** 192.168.57.105

Click on a "**Username**" in the table to edit

Username	Surname	Email	Home Directory	Login Shell
dilan	asanga		/home	/bin/bash
keneth	perera	keneth@gmail.com	/home/keneth	/bin/bash
neo	silva	neo@gmail.com	/home/neo	/bin/bash
kasun	silva	kasun@gmail.com	/home/kasun	/bin/bash
prasad	wimalasena	prasad@gmail.com	/home/prasad	/bin/bash
test_password	test_user	test@gmail.com	/home/test	/bin/bash
test_key	test_key	test@gmail.com	/home/test_key	/bin/bash
manula	silva	manula@gmail.com	/home/manula	/bin/bash
nipuna	silva	nipuna@gmail.com	/home/nipuna	/bin/bash
ashwin	ashwin	ashwin@gmail.com	/home/ashwin	/bin/bash
test123	test123	test123@gmail.com	/home/test123	/bin/bash
shehan	perera	shehan@gmail.com	/home/shehan	/bin/bash
rasanjaya	silva	rasanjaya@gmail.com	/home/rasanjaya	/bin/bash
anura	perera	anura@gmail.com	/home/anura	/bin/bash

General

[Home](#)

Groups

[List All Groups](#)

[Create Group](#)

[Delete Group](#)

[Add Users To Group](#)

[List Users In Group](#)

[Delete Users From Group](#)

Users

[Search Users](#)

[Create User](#)

[Edit User](#)

[Delete User](#)

[Import User](#)

Servers

[Add LDAP Server](#)

[Modify LDAP Server](#)

[Delete LDAP Server](#)

[Add/Edit Base Server](#)

Figure 4.2.6

7). View users in each group.

The screenshot shows a web browser window with the address bar displaying "https://192.168.57.105/lim/groups/viewgroups.php". The page title is "LIM". Below the title, it says "Displaying Users In Selected Group". The user information is "Username: admin DN: ou=users,dc=ucsc,dc=org Server: 192.168.57.105".

On the left side, there is a navigation menu with the following sections:

- General**
 - [Home](#)
- Groups**
 - [List All Groups](#)
 - [Create Group](#)
 - [Delete Group](#)
 - [Add Users To Group](#)
 - [List Users In Group](#)
 - [Delete Users From Group](#)
- Users**
 - [Search Users](#)

The main content area is titled "Group Members" and displays a list of users in a table:

Group Members
manula
nipuna
ashwin
rasanjaya
anura
test01
test02

Figure 4.2.7

Appendix II: Output of the LDAP search query

By running the below command, the whole retailed entry for a given user can be retrieved.

Command.

```
ldapsearch -x -h $server -p $port -o ldif-wrap=no -b $basedn -s sub
"(&(objectClass=posixAccount)(uid=$cn))"
```

Output.

```
# extended LDIF
#
# LDAPv3
# base <ou=users,dc=ucsc,dc=org> with scope subtree
# filter: (&(objectClass=posixAccount)(uid=dilan))
# requesting: ALL
#
# dilan asanga, users, ucsc.org
dn: cn=dilan asanga,ou=users,dc=ucsc,dc=org
cn: dilan asanga
givenName: dilan
gidNumber: 500
homeDirectory: /home/users/dilan
sn: asanga
loginShell: /bin/bash
objectClass: inetOrgPerson
```

```

objectClass: posixAccount
objectClass: top
objectClass: ldapPublicKey
uidNumber: 1000
uid: dilan
sshPublicKey: ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAQFmNfqeJ6H8B1iWXRaKf9qqgp5PxUeKGUcQ/ldeE7bYIQ
z96+mxOGyED9Yz3D93/SEGZU2QynuBssU+mhts/y7NVL2tVRldBiQtabwbCqtQL7JpgWqAlfCcm
jNbyQVyzxWj53GEh7Hy14qS7eF2KT1+DahrL15v+wm+cUGK0KBPvav0ivO9ePBbG+Md3n6IDoN2
opWm40JI7aqqN7dr3lwnVXODt1Y/wOzypCDlozN6dNuYv/zzTVB+FDtURbGi4YJiQsR89HybVRp
uEO2pRctENggW4symZiwtsI6c0kvDoc/XJOWPlp6IW4JOrBJPS+FOTQBjed6e9Ut8/rJekxZ
dilang@wso2.com

# search result
search: 2
result: 0 Success

# numResponses: 2
# numEntries: 1

```

Appendix III: LDAP Entry for the user “anura”

```

# LDIF Export for cn=anura prera,ou=users,dc=ucsc,dc=org
# Server: My LDAP Server (192.168.57.105)
# Search Scope: base
# Search Filter: (objectClass=*)
# Total Entries: 1
#
# Generated by phpLDAPadmin (http://phpldapadmin.sourceforge.net) on March 7, 2017
11:52 pm
# Version: 1.2.2

version: 1

# Entry 1: cn=anura prera,ou=users,dc=ucsc,dc=org
dn: cn=anura prera,ou=users,dc=ucsc,dc=org
cn: anura prera
gidnumber: 501
givenname: anura
homedirectory: /home/users/anura
loginshell: /bin/bash
objectclass: inetOrgPerson
objectclass: posixAccount
objectclass: top
objectclass: ldapPublicKey
sn: prera
sshpublickey: ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAQCEYI1Y2w8C+X0ySu5NHnKnN
CwtInr6EA5VOjL08TrtvC1L6rWHTaN3XJ6bDBHiIpYvAJmDylbKfRmjtyFpSeMvCWAhCcaplbr/
rhJ3jDfqVd4WagRzz3O0VidXHeZq7BT8NFd+DtnLpOfdQ1kF8jTLuXEU4ctR8U3rQvdpdpcT3o1b
ieAmrx2J0EjTxyzCd9pTnsTFUed4ChjrKUnS+st491d8GPRUIiiz4og1KDO8QSX5/LzbY8o6oX
rEdL/YpEKD5g8VfZBjpeEeChy7DDI4t3ccj5Sceg1ujl+EKJxppfxfvdsSJ97o43dyAQ3dD014J
lznaEXuWxz/EkWVuJx dilan@Dilans-MacBook-Pro.local
uid: anura
uidnumber: 1003
userpassword: {MD5}5ydD8a9T/b2EDx/dP9jvEA==

```

Appendix IV: SSH Client verbose output

```
dilan@Dilans-MacBook-Pro:~$ ssh -v anura@192.168.57.102 -i /tmp/id_rsa_anura
OpenSSH_6.2p2, OpenSSL 0.9.8r 8 Dec 2011
debug1: Reading configuration data /Users/dilan/.ssh/config
debug1: Reading configuration data /etc/ssh_config
debug1: /etc/ssh_config line 20: Applying options for *
debug1: /etc/ssh_config line 102: Applying options for *
debug1: Connecting to 192.168.57.102 [192.168.57.102] port 22.
debug1: Connection established.
debug1: identity file /tmp/id_rsa_anura type 1
debug1: identity file /tmp/id_rsa_anura-cert type -1
debug1: Enabling compatibility mode for protocol 2.0
debug1: Local version string SSH-2.0-OpenSSH_6.2
debug1: Remote protocol version 2.0, remote software version OpenSSH_6.6.1p1
Ubuntu-2ubuntu2.8
debug1: match: OpenSSH_6.6.1p1 Ubuntu-2ubuntu2.8 pat OpenSSH*
debug1: SSH2_MSG_KEXINIT sent
debug1: SSH2_MSG_KEXINIT received
debug1: kex: server->client aes128-ctr hmac-md5-etm@openssh.com none
debug1: kex: client->server aes128-ctr hmac-md5-etm@openssh.com none
debug1: SSH2_MSG_KEX_DH_GEX_REQUEST(1024<1024<8192) sent
debug1: expecting SSH2_MSG_KEX_DH_GEX_GROUP
debug1: SSH2_MSG_KEX_DH_GEX_INIT sent
debug1: expecting SSH2_MSG_KEX_DH_GEX_REPLY
debug1: Server host key: RSA a1:53:b1:71:f4:dc:50:51:ed:36:6e:01:ea:93:c3:8c
debug1: Host '192.168.57.102' is known and matches the RSA host key.
debug1: Found key in /Users/dilan/.ssh/known_hosts:72
debug1: ssh_rsa_verify: signature correct
debug1: SSH2_MSG_NEWKEYS sent
debug1: expecting SSH2_MSG_NEWKEYS
debug1: SSH2_MSG_NEWKEYS received
debug1: Roaming not allowed by server
debug1: SSH2_MSG_SERVICE_REQUEST sent
debug1: SSH2_MSG_SERVICE_ACCEPT received
debug1: Authentications that can continue: publickey,password
debug1: Next authentication method: publickey
debug1: Offering RSA public key: /tmp/id_rsa_anura
debug1: Server accepts key: pkalg ssh-rsa blen 279
debug1: read PEM private key done: type RSA
debug1: Authentication succeeded (publickey).
Authenticated to 192.168.57.102 ([192.168.57.102]:22).
debug1: channel 0: new [client-session]
debug1: Requesting no-more-sessions@openssh.com
debug1: Entering interactive session.
debug1: Requesting authentication agent forwarding.
debug1: Sending environment.
debug1: Sending env LC_ALL = en_US.UTF-8
debug1: Sending env LC_CTYPE = en_US.UTF-8
Welcome to Ubuntu 14.04.5 LTS (GNU/Linux 3.19.0-64-generic x86_64)
```

* Documentation: <https://help.ubuntu.com/>

System information as of Wed Mar 8 05:53:39 +0530 2017

System load:	0.0	Users logged in:	1
Usage of /:	62.5% of 7.26GB	IP address for eth0:	10.0.2.15
Memory usage:	14%	IP address for eth1:	10.10.10.20
Swap usage:	0%	IP address for eth2:	192.168.57.102
Processes:	96		

Graph this data and manage this system at:
<https://landscape.canonical.com/>

WARNING: Security updates for your current Hardware Enablement Stack ended on 2016-08-04:

* http://wiki.ubuntu.com/1404_HWE_EOL

To upgrade to a supported (or longer-supported) configuration:

* Upgrade from Ubuntu 14.04 LTS to Ubuntu 16.04 LTS by running:
sudo do-release-upgrade

OR

* Switch to the current security-supported stack by running:
sudo apt-get install linux-image-generic-lts-xenial linux-generic-lts-xenial

and reboot your system.

Last login: Wed Mar 8 05:53:39 2017 from 192.168.57.1
anura@bastion:~\$

Appendix V: LDAP debug Log

```
58bf458d => access_allowed: read access to "cn=anura prera,ou=users,dc=ucsc,dc=org"
"userPassword" requested
58bf458d => acl_get: [1] attr userPassword
58bf458d => acl_mask: access to entry "cn=anura prera,ou=users,dc=ucsc,dc=org", attr
"userPassword" requested
58bf458d => acl_mask: to value by "", (=0)
58bf458d <= check_a_dn_pat: self
58bf458d <= check_a_dn_pat: anonymous
58bf458d <= acl_mask: [2] applying auth(=xd) (stop)
58bf458d <= acl_mask: [2] mask: auth(=xd)
58bf458d => slap_access_allowed: read access denied by auth(=xd)
58bf458d => access_allowed: no more rules
58bf458d send_search_entry: conn 1014 access to attribute userPassword, value #0 not allowed
58bf458d => access_allowed: result not in cache (sshPublicKey)
58bf458d => access_allowed: read access to "cn=anura prera,ou=users,dc=ucsc,dc=org"
"sshPublicKey" requested
58bf458d => dn: [2]
58bf458d => acl_get: [3] attr sshPublicKey
58bf458d => acl_mask: access to entry "cn=anura prera,ou=users,dc=ucsc,dc=org", attr
"sshPublicKey" requested
58bf458d => acl_mask: to value by "", (=0)
58bf458d <= check_a_dn_pat: cn=admin,dc=ucsc,dc=org
58bf458d <= check_a_dn_pat: *
58bf458d <= acl_mask: [2] applying read(=rscxd) (stop)
58bf458d <= acl_mask: [2] mask: read(=rscxd)
58bf458d => slap_access_allowed: read access granted by read(=rscxd)
58bf458d => access_allowed: read access granted by read(=rscxd)
58bf458d conn=1014 op=1 ENTRY dn="cn=anura prera,ou=users,dc=ucsc,dc=org"
ber_flush2: 738 bytes to sd 18
ldap_write: want=738, written=738
0000: 30 82 02 de 02 01 02 64 82 02 d7 04 26 63 6e 3d 0.....d....&cn=
0010: 61 6e 75 72 61 20 70 72 65 72 61 2c 6f 75 3d 75 anura prera,ou=u
0020: 73 65 72 73 2c 64 63 3d 75 63 73 63 2c 64 63 3d sers,dc=ucsc,dc=
0030: 6f 72 67 30 82 02 ab 30 13 04 02 63 6e 31 0d 04 org0...0...cn1..
0040: 0b 61 6e 75 72 61 20 70 72 65 72 61 30 12 04 09 .anura prera0...
0050: 67 69 64 4e 75 6d 62 65 72 31 05 04 03 35 30 31 gidNumber1...501
0060: 30 14 04 09 67 69 76 65 6e 4e 61 6d 65 31 07 04 0...givenName1..
0070: 05 61 6e 75 72 61 30 24 04 0d 68 6f 6d 65 44 69 .anura0$.homeDi
0080: 72 65 63 74 6f 72 79 31 13 04 11 2f 68 6f 6d 65 rectory1.../home
0090: 2f 75 73 65 72 73 2f 61 6e 75 72 61 30 19 04 0a /users/anura0...
00a0: 6c 6f 67 69 6e 53 68 65 6c 6c 31 0b 04 09 2f 62 loginShell1.../b
00b0: 69 6e 2f 62 61 73 68 30 40 04 0b 6f 62 6a 65 63 in/bash0@..objec
00c0: 74 43 6c 61 73 73 31 31 04 0d 69 6e 65 74 4f 72 tClass11..inetOr
00d0: 67 50 65 72 73 6f 6e 04 0c 70 6f 73 69 78 41 63 gPerson..posixAc
00e0: 63 6f 75 6e 74 04 03 74 6f 70 04 0d 6c 64 61 70 count..top..ldap
00f0: 50 75 62 6c 69 63 4b 65 79 30 0d 04 02 73 6e 31 PublicKey0...sn1
0100: 07 04 05 70 72 65 72 61 30 0e 04 03 75 69 64 31 ...prera0...uid1
0110: 07 04 05 61 6e 75 72 61 30 13 04 09 75 69 64 4e ...anura0...uidN
0120: 75 6d 62 65 72 31 06 04 04 31 30 30 33 30 82 01 umber1...10030..
0130: b1 04 0c 73 73 68 50 75 62 6c 69 63 4b 65 79 31 ...sshPublicKey1
0140: 82 01 9f 04 82 01 9b 73 73 68 2d 72 73 61 20 41 .....ssh-rsa A
0150: 41 41 41 42 33 4e 7a 61 43 31 79 63 32 45 41 41 AAAB3NzaC1yc2EAA
0160: 41 41 44 41 51 41 42 41 41 41 42 41 51 43 78 65 AADAQAABAABACxe
-----
02b0: 4a 6c 7a 6e 61 45 58 75 57 78 7a 2f 45 6b 57 56 J1znaEXuWxz/EkWW
02c0: 75 4a 78 20 64 69 6c 61 6e 40 44 69 6c 61 6e 73 uJx dilan@Dilans
02d0: 2d 4d 61 63 42 6f 6f 6b 2d 50 72 6f 2e 6c 6f 63 -MacBook-Pro.loc
02e0: 61 6c al
58bf458d <= send_search_entry: conn 1014 exit.
```

Appendix VI: SSH Client Log

```

dilan@Dilans-MacBook-Pro:~$ ssh -v anura@192.168.57.102
OpenSSH_6.2p2, OpenSSL 0.9.8r 8 Dec 2011
debug1: Reading configuration data /Users/dilan/.ssh/config
debug1: Reading configuration data /etc/ssh_config
debug1: /etc/ssh_config line 20: Applying options for *
debug1: /etc/ssh_config line 102: Applying options for *
debug1: Connecting to 192.168.57.102 [192.168.57.102] port 22.
debug1: Connection established.
debug1: identity file /Users/dilan/.ssh/id_rsa type 1
debug1: identity file /Users/dilan/.ssh/id_rsa-cert type -1
debug1: identity file /Users/dilan/.ssh/id_dsa type -1
debug1: identity file /Users/dilan/.ssh/id_dsa-cert type -1
debug1: Enabling compatibility mode for protocol 2.0
debug1: Local version string SSH-2.0-OpenSSH_6.2
debug1: Remote protocol version 2.0, remote software version OpenSSH_6.6.1p1
Ubuntu-2ubuntu2.8
debug1: match: OpenSSH_6.6.1p1 Ubuntu-2ubuntu2.8 pat OpenSSH*
debug1: SSH2_MSG_KEXINIT sent
debug1: SSH2_MSG_KEXINIT received
debug1: kex: server->client aes128-ctr hmac-md5-etm@openssh.com none
debug1: kex: client->server aes128-ctr hmac-md5-etm@openssh.com none
debug1: SSH2_MSG_KEX_DH_GEX_REQUEST(1024<1024<8192) sent
debug1: expecting SSH2_MSG_KEX_DH_GEX_GROUP
debug1: SSH2_MSG_KEX_DH_GEX_INIT sent
debug1: expecting SSH2_MSG_KEX_DH_GEX_REPLY
debug1: Server host key: RSA a1:53:b1:71:f4:dc:50:51:ed:36:6e:01:ea:93:c3:8c
debug1: Host '192.168.57.102' is known and matches the RSA host key.
debug1: Found key in /Users/dilan/.ssh/known_hosts:72
debug1: ssh_rsa_verify: signature correct
debug1: SSH2_MSG_NEWKEYS sent
debug1: expecting SSH2_MSG_NEWKEYS
debug1: SSH2_MSG_NEWKEYS received
debug1: Roaming not allowed by server
debug1: SSH2_MSG_SERVICE_REQUEST sent
debug1: SSH2_MSG_SERVICE_ACCEPT received
debug1: Authentications that can continue: publickey,password
debug1: Next authentication method: publickey
debug1: Offering RSA public key: /tmp/id_rsa_anura
debug1: Server accepts key: pkalg ssh-rsa blen 279
debug1: Authentication succeeded (publickey).
Authenticated to 192.168.57.102 ([192.168.57.102]:22).
debug1: channel 0: new [client-session]
debug1: Requesting no-more-sessions@openssh.com
debug1: Entering interactive session.
debug1: Requesting authentication agent forwarding.
debug1: Sending environment.
debug1: Sending env LC_ALL = en_US.UTF-8
debug1: Sending env LC_CTYPE = en_US.UTF-8
Welcome to Ubuntu 14.04.5 LTS (GNU/Linux 3.19.0-64-generic x86_64)

* Documentation:  https://help.ubuntu.com/

System information as of Wed Mar  8 16:49:10 +0530 2017

System load:  0.0                Users logged in:    1
Usage of /:   62.5% of 7.26GB     IP address for eth0: 10.0.2.15
Memory usage: 14%                IP address for eth1: 10.10.10.20
Swap usage:   0%                 IP address for eth2: 192.168.57.102
Processes:   94

Last login: Wed Mar  8 16:45:04 2017 from 192.168.57.1
anura@bastion:~$
anura@bastion:~$
anura@bastion:~$ ssh -v anura@192.168.57.104
OpenSSH_6.6.1, OpenSSL 1.0.1f 6 Jan 2014
debug1: Reading configuration data /etc/ssh/ssh_config
debug1: /etc/ssh/ssh_config line 19: Applying options for *
debug1: Connecting to 192.168.57.104 [192.168.57.104] port 22.

```

```

debug1: Connection established.
debug1: identity file /home/users/anura/.ssh/id_rsa type -1
debug1: identity file /home/users/anura/.ssh/id_rsa-cert type -1
debug1: identity file /home/users/anura/.ssh/id_dsa type -1
debug1: identity file /home/users/anura/.ssh/id_dsa-cert type -1
debug1: identity file /home/users/anura/.ssh/id_ecdsa type -1
debug1: identity file /home/users/anura/.ssh/id_ecdsa-cert type -1
debug1: identity file /home/users/anura/.ssh/id_ed25519 type -1
debug1: identity file /home/users/anura/.ssh/id_ed25519-cert type -1
debug1: Enabling compatibility mode for protocol 2.0
debug1: Local version string SSH-2.0-OpenSSH_6.6.1p1 Ubuntu-2ubuntu2.8
debug1: Remote protocol version 2.0, remote software version OpenSSH_6.6.1p1
Ubuntu-2ubuntu2.8
debug1: match: OpenSSH_6.6.1p1 Ubuntu-2ubuntu2.8 pat OpenSSH_6.6.1* compat
0x04000000
debug1: SSH2_MSG_KEXINIT sent
debug1: SSH2_MSG_KEXINIT received
debug1: kex: server->client aes128-ctr hmac-md5-etm@openssh.com none
debug1: kex: client->server aes128-ctr hmac-md5-etm@openssh.com none
debug1: sending SSH2_MSG_KEX_ECDH_INIT
debug1: expecting SSH2_MSG_KEX_ECDH_REPLY
debug1: Server host key: ECDSA d8:a6:97:0d:3e:55:76:53:03:a7:07:45:a1:b7:fa:ff
debug1: Host '192.168.57.104' is known and matches the ECDSA host key.
debug1: Found key in /home/users/anura/.ssh/known_hosts:1
debug1: ssh_ecdsa_verify: signature correct
debug1: SSH2_MSG_NEWKEYS sent
debug1: expecting SSH2_MSG_NEWKEYS
debug1: SSH2_MSG_NEWKEYS received
debug1: SSH2_MSG_SERVICE_REQUEST sent
debug1: SSH2_MSG_SERVICE_ACCEPT received
debug1: client_input_channel_open: ctype auth-agent@openssh.com rchan 2 win 65536
max 16384
debug1: channel 1: new [authentication agent connection]
debug1: confirm auth-agent@openssh.com
debug1: Authentications that can continue: publickey,password
debug1: Next authentication method: publickey
debug1: Offering RSA public key: /tmp/id_rsa_anura
debug1: Server accepts key: pkalg ssh-rsa blen 279
debug1: channel 1: FORCE input drain
debug1: Authentication succeeded (publickey).
Authenticated to 192.168.57.104 ([192.168.57.104]:22).
debug1: channel 0: new [client-session]
debug1: Requesting no-more-sessions@openssh.com
debug1: Entering interactive session.
debug1: channel 1: free: authentication agent connection, nchannels 2
debug1: Requesting authentication agent forwarding.
debug1: Sending environment.
debug1: Sending env LC_ALL = en_US.UTF-8
debug1: Sending env LANG = en_US.UTF-8
debug1: Sending env LC_CTYPE = en_US.UTF-8
Welcome to Ubuntu 14.04.4 LTS (GNU/Linux 4.2.0-27-generic x86_64)

* Documentation:  https://help.ubuntu.com/

System information as of Wed Mar  8 17:00:54 IST 2017

System load:  0.0                Users logged in:  1
Usage of /:   30.0% of 7.75GB    IP address for eth0: 10.0.2.15
Memory usage: 5%                IP address for eth1: 10.10.10.4
Swap usage:   0%                IP address for eth2: 192.168.57.104
Processes:   88

Last login: Wed Mar  8 17:00:54 2017 from 192.168.57.102
anura@sshserver:~$

```

Appendix VII: Creating a new RSA key pair

```

Last login: Wed Mar  8 17:01:14 on ttys006
dilan@Dilans-MacBook-Pro:~$ ssh-keygen -t rsa -C "charith@email.com"
Generating public/private rsa key pair.
Enter file in which to save the key (/Users/dilan/.ssh/id_rsa): /tmp/id_rsa_charith
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /tmp/id_rsa_charith.
Your public key has been saved in /tmp/id_rsa_charith.pub.
The key fingerprint is:
6f:5e:c9:a6:8f:47:3f:02:b8:c6:fa:fe:08:df:10:c5 charith@email.com
The key's randomart image is:
+--[ RSA 2048]-----+
|
|      .
|      E
|      .
|      S.
|      .o....
|      ...oo=.
|      o+* =o o
|      ,=. *o. .
+-----+
dilan@Dilans-MacBook-Pro:~$ cat /tmp/id_rsa_charith
-----BEGIN RSA PRIVATE KEY-----
MIIEpAIBAAKCAQEASd/r9qCokIcfcKn+6+qc/brYPqWpkSfB3xkj0eJ/MUNKoNhv
8KRave/GwxloiITd+6+oiqSmrz5CyuTdWovKE29x1KXGzo1WFSU33HM6GZQsINNv
4f4J9lPwyMA2Z0SEf7d8p5tG8sKfnVXuD1AiSzUDfemJieJkpoWhZsmX0iwdmATd
HVNK7X8ebVtH3qLUuN0RGRF1wMnbSsz5GMXy9veZpE3GjuwHunH2imN1jBLdaYvi
jF3k032K7rdzK8ux68wdnEb2FRdb4RmSx7wmbRShfFNwCp2NCmq200s8E7GK0pYH
8E6dHAWmX+4KdqpK1UECq3A8NiQSLH1NKyR+QIDAQABAoIBAQQDfDzmQgTdwNzU
4RqJfSoW+WyokABt0BD9DSVesD/QP/9Zxe50qZ9gz+BMiAuc3rz6L6BsQ5910
IRa52qkJ05/5/F0SLVpHeorfBjRIgT2Ue3WbnrRYgvp4bEF2iaiz7s6Cr/N8Ma2v
6MS0pjtWGIzY0fyfd2R8v0/SmEHsHsCr2jx6Sk3GF+X6Zk09M2igu1BEq0gd323
2qj+S2KtA1KKYqMP5YNbCfBIzh0Mji04t/ZU2vVzdIk9i3pHiG4MEv3ADap1SC
IqvhX3sxYlgy9mU2x5chCCLBgdhqj7aMf2zVwha6XFfsgNxEof+zP1tphgxcXBI1
RvIdHyHxAoGBAPp7/9RCbc9vXSLM/Nvb5U4FHUxiqJmJ7JjKN85PJZIRL6iRxtAC
9IHUoatGwt9RUeCiPbI6vam4zuGGvqWZfsFKCrVva/75iqs0XEztAq+jHvbih/ak
D7ICB1Q7UFkNosERJzbnV9k0r2Fa1//QEVCFLoIqC5A2AKSLmJeQBLAoGBA0rv
vnVD6PmAQALozH5zgeKaoIoDdq+/4vaSvHwqzTK0BSNHJlAASPrQIGe8M2jhqK
V0iGRNzXrnALtt8rZrT14B1zS0hX9+90gTLfIZuYgwqFa2MaJ4J0M5PJlGc5vbWx
+ncX10xk1iDx1fMKGi5wZ3U0N9/d6rNKZGyKlVRLAoGASL6kUKeuLnU+YLn4AX3+
PUddbTQBtP2KWeE4jdEMjzwwHbs6CwUgsUsq3IwYinCxaM3mcBXfqfuzS71xxNO
0Xpf5NZ5k8YzEH+XYrvecNXF01t8sxN4f3D5F5NWRVahcsKHi5omfpampTmqzC2E
c8EBb6sdC2JP6Z9zckdrfbsCgYEAmSkEzGP9DGe0/TeZA8qsQ4Q8XV919GL9tzn
XBBV3N/sUiGlaXKstLSIYJtS5vAZAjpWqarSGAv/dKpeZ01M9bJrP4x73eR/mP6
nJSUZ2BFBcXnNwtmmG+s9kr0TLLKd7p448zwaPAmY4HljhVm7iF/gQwy5PJcWPHd
uFk9gjMcgYBYAR09PUEHDMuLtv0GuWb7JNuARaxMiWrnd+cYG8MZ3xKteVsNHQXA
k8ta/zvuf/9prcFE8QPbE4Vmqdh/2A6KltDTLIEaBCPPvWzohVtGkzjSmMNGkeFb
zRiNzNhKqm0kKW850k1Nq1iJh1Kx+DDw6JIhyTF7IskY2Es0f8IyVw==
-----END RSA PRIVATE KEY-----
dilan@Dilans-MacBook-Pro:~$ cat /tmp/id_rsa_charith.pub
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAQDl3+v2oKgohx9ySf7r6pz9utg+pamRJ8HFGSPR4n8xQ0qg2G/wpFq978bdGWiIhN37r6iKpKavPkLK611a
i8oTb3GUpcb0jvYXLTFcczoZlCwg0Z/h/gn2WnDIwDznRIR/t3ynm0byw+p+dVe4PUCJLNQN94wmJ4mSmhaFmyZfSLB2YBN0dU0rtfx5tW0feqV543REZEXXA
w1tKzPkYxFL295mkTca07Ae6cfaKY3WMEt1pi+KMxeQ7fYrut3Mry7HrzB2cRvYVF1vhGZLHVDBtFKF8U3AKN0KarBTszwTsyRSlgfwTp0cBabF7go0qkr
VQQKrcDw2JBKUfU0rJH5 charith@email.com
dilan@Dilans-MacBook-Pro:~$

```

Figure 5.1.4 Creating a new RSA key pair

Appendix VIII: Adding a user to the LDAP Server

1. LDAP schema of an existing user

```
# LDIF Export for cn=anura prera,ou=users,dc=ucsc,dc=org
# Server: My LDAP Server (192.168.57.105)
# Search Scope: base
# Search Filter: (objectClass=*)
# Total Entries: 1
#
# Generated by phpLDAPadmin (http://phpldapadmin.sourceforge.net) on March 8, 2017 4:55 am
# Version: 1.2.2

version: 1

# Entry 1: cn=anura prera,ou=users,dc=ucsc,dc=org
dn: cn=anura prera,ou=users,dc=ucsc,dc=org
cn: anura prera
gidnumber: 501
givenname: anura
homedirectory: /home/users/anura
loginshell: /bin/bash
objectclass: inetOrgPerson
objectclass: posixAccount
objectclass: top
objectclass: ldapPublicKey
sn: prera
sshpublickey: ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAQACxeYI1Y2w8C+X0ySu5NHNKnN
CwtInr6EA5VOjL08TrtvC1L6rWHTaN3XJ6bDBHiIpYvAJmDylbKfRmjtyFpSeMvCWAhCcaplbr/
rhJ3jDfJqVd4WagRzz300VidXHeZq7BT8NFD+DtnLpOfdQlkF8jTLuXEU4ctr8U3rQvdpct3o1b
ieAmrx2J0EjTxxzCd9pTnsTFUed4ChjrkUnS+st491d8GPRUIiiz4og1KDO8QSX5/LzbY8o6oX
rEdL/YpEKD5g8VfzBgjpeEChy7DDI4t3ccj5Scegluj1+EKJxppfxFVdsSj97o43dyAQ3dD014J
lznaEXuWxz/EkWVuJx dilan@Dilans-MacBook-Pro.local
uid: anura
uidnumber: 1003
userpassword: {MD5}5ydD8a9T/b2EDx/dP9jvEA==
```

Figure 5.1.5 Existing user LDAP schema

2. Importing a new user to LDAP server

Import

Server: My LDAP Server

Select an LDIF file Choose File No file chosen

Maximum file size 2M

Or paste your LDIF here

```
# LDIF Export for cn=charith kangara,ou=users,dc=ucsc,dc=org
# Server: My LDAP Server (192.168.57.105)
# Search Scope: base
# Search Filter: (objectClass=*)
# Total Entries: 1
#
# Generated by phpLDAPadmin (http://phpldapadmin.sourceforge.net) on March 8, 2017 4:55 am
# Version: 1.2.2

version: 1

# Entry 1: cn=charith kangara,ou=users,dc=ucsc,dc=org
dn: cn=charith kangara,ou=users,dc=ucsc,dc=org
cn: charith kangara
gidnumber: 501
givenname: charith
homedirectory: /home/users/charith
loginshell: /bin/bash
objectclass: inetOrgPerson
objectclass: posixAccount
objectclass: top
objectclass: ldapPublicKey
sn: kangara
ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAQADQI3+v2oKgoHx9ySf7r6pz9utg+nAmRI8HfGSPR4n8xQ0gg2G/wpFq978bDGWihN37r6iKpKavPkLk611ai8o
Tb3GUnchQjVYXlJfccoZlCwg02/h/gn2WnDlwDznRlR/t3ynm0bywp+dVe4PUCJLNQN94wmj4mSmhaFmyZfSLB2Y8N0dU0rtfxStW0feqV543REZEXX
Aw1tKzPkyxfl295mkTcaO7Ae6cfaKY3WMEt1pi+KMxeQ7fyrut3Mry7HrzB2crvYVf1vhGZLhVDBfKF8U3AKnY0KarbtSzwTsYrSglfwTp0cBabFf7goO
qkrVQQKrcDw2JBKUFU0rJH5 charith@email.com
uid: charith
uidnumber: 1004|
userpassword: {MD5}5ydD8a9T/b2EDx/dP9jvEA==
```

Don't stop on errors

Figure 5.1.6 Adding a new user to LDAP

3. New user is added to the LDAP server



Figure 5.1.7 New user added to LDAP successfully

4. New user is in LDAP schema

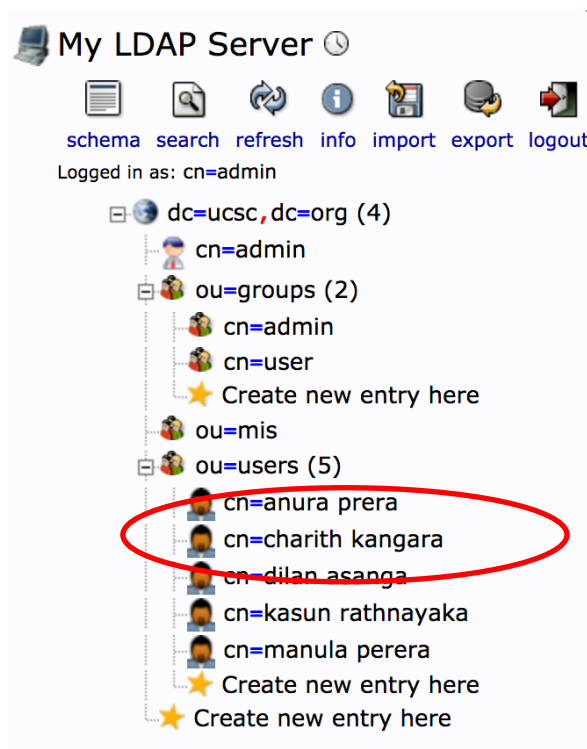


Figure 5.1.8 New user in LDAP user store

Appendix IX: Login as a user to the bastion server

1. SSH Client debug output without Private Key provided

```
dilan@Dilans-MacBook-Pro:~$ ssh -v charith@192.168.57.102
OpenSSH_6.2p2, OpenSSL 0.9.8r 8 Dec 2011
debug1: Reading configuration data /Users/dilan/.ssh/config
debug1: Reading configuration data /etc/ssh_config
debug1: /etc/ssh_config line 20: Applying options for *
debug1: /etc/ssh_config line 102: Applying options for *
debug1: Connecting to 192.168.57.102 [192.168.57.102] port 22.
debug1: Connection established.
debug1: identity file /Users/dilan/.ssh/id_rsa type 1
debug1: identity file /Users/dilan/.ssh/id_rsa-cert type -1
debug1: identity file /Users/dilan/.ssh/id_dsa type -1
debug1: identity file /Users/dilan/.ssh/id_dsa-cert type -1
debug1: Enabling compatibility mode for protocol 2.0
debug1: Local version string SSH-2.0-OpenSSH_6.2
debug1: Remote protocol version 2.0, remote software version OpenSSH_6.6.1p1 Ubuntu-2ubuntu2.8
debug1: match: OpenSSH_6.6.1p1 Ubuntu-2ubuntu2.8 pat OpenSSH*
debug1: SSH2_MSG_KEXINIT sent
```

```

debug1: SSH2_MSG_KEXINIT received
debug1: kex: server->client aes128-ctr hmac-md5-etm@openssh.com none
debug1: kex: client->server aes128-ctr hmac-md5-etm@openssh.com none
debug1: SSH2_MSG_KEX_DH_GEX_REQUEST(1024<1024<8192) sent
debug1: expecting SSH2_MSG_KEX_DH_GEX_GROUP
debug1: SSH2_MSG_KEX_DH_GEX_INIT sent
debug1: expecting SSH2_MSG_KEX_DH_GEX_REPLY
debug1: Server host key: RSA a1:53:b1:71:f4:dc:50:51:ed:36:6e:01:ea:93:c3:8c
debug1: Host '192.168.57.102' is known and matches the RSA host key.
debug1: Found key in /Users/dilan/.ssh/known_hosts:72
debug1: ssh_rsa_verify: signature correct
debug1: SSH2_MSG_NEWKEYS sent
debug1: expecting SSH2_MSG_NEWKEYS
debug1: SSH2_MSG_NEWKEYS received
debug1: Roaming not allowed by server
debug1: SSH2_MSG_SERVICE_REQUEST sent
debug1: SSH2_MSG_SERVICE_ACCEPT received
debug1: Authentications that can continue: publickey,password
debug1: Next authentication method: publickey
debug1: Offering RSA public key: /tmp/id_rsa_anura
debug1: Authentications that can continue: publickey,password
debug1: Offering RSA public key: /Users/dilan/.ssh/id_rsa
debug1: Authentications that can continue: publickey,password
debug1: Trying private key: /Users/dilan/.ssh/id_dsa
debug1: Next authentication method: password
charith@192.168.57.102's password:

```

2. SSH Client debug output with Private key provided

```

dilan@Dilans-MacBook-Pro:~$ ssh -v charith@192.168.57.102 -i /tmp/id_rsa_charith
OpenSSH_6.2p2, OpenSSL 0.9.8r 8 Dec 2011
debug1: Reading configuration data /Users/dilan/.ssh/config
debug1: Reading configuration data /etc/ssh_config
debug1: /etc/ssh_config line 20: Applying options for *
debug1: /etc/ssh_config line 102: Applying options for *
debug1: Connecting to 192.168.57.102 [192.168.57.102] port 22.
debug1: Connection established.
debug1: identity file /tmp/id_rsa_charith type 1
debug1: identity file /tmp/id_rsa_charith-cert type -1
debug1: Enabling compatibility mode for protocol 2.0
debug1: Local version string SSH-2.0-OpenSSH_6.2
debug1: Remote protocol version 2.0, remote software version OpenSSH_6.6.1p1 Ubuntu-2ubuntu2.8
debug1: match: OpenSSH_6.6.1p1 Ubuntu-2ubuntu2.8 pat OpenSSH*
debug1: SSH2_MSG_KEXINIT sent
debug1: SSH2_MSG_KEXINIT received
debug1: kex: server->client aes128-ctr hmac-md5-etm@openssh.com none
-----
debug1: Server host key: RSA a1:53:b1:71:f4:dc:50:51:ed:36:6e:01:ea:93:c3:8c
debug1: Host '192.168.57.102' is known and matches the RSA host key.
debug1: Found key in /Users/dilan/.ssh/known_hosts:72
debug1: ssh_rsa_verify: signature correct
debug1: SSH2_MSG_NEWKEYS sent
debug1: expecting SSH2_MSG_NEWKEYS
debug1: SSH2_MSG_NEWKEYS received
debug1: Roaming not allowed by server
debug1: SSH2_MSG_SERVICE_REQUEST sent
debug1: SSH2_MSG_SERVICE_ACCEPT received
debug1: Authentications that can continue: publickey,password
debug1: Next authentication method: publickey
debug1: Offering RSA public key: /tmp/id_rsa_anura
debug1: Authentications that can continue: publickey,password
debug1: Offering RSA public key: /tmp/id_rsa_charith
debug1: Server accepts key: pka1g ssh-rsa blen 279
debug1: read PEM private key done: type RSA
debug1: Authentication succeeded (publickey).
Authenticated to 192.168.57.102 ([192.168.57.102]:22).
debug1: channel 0: new [client-session]
debug1: Requesting no-more-sessions@openssh.com
debug1: Entering interactive session.
debug1: Requesting authentication agent forwarding.
debug1: Sending environment.
debug1: Sending env LC_ALL = en_US.UTF-8
debug1: Sending env LC_CTYPE = en_US.UTF-8
Creating directory '/home/users/charith'.
Welcome to Ubuntu 14.04.5 LTS (GNU/Linux 3.19.0-64-generic x86_64)

```

* Documentation: <https://help.ubuntu.com/>

System information as of Wed Mar 8 17:01:47 +0530 2017

```
System load: 0.0          Users logged in: 2
Usage of /: 62.5% of 7.26GB  IP address for eth0: 10.0.2.15
Memory usage: 14%         IP address for eth1: 10.10.10.20
Swap usage: 0%           IP address for eth2: 192.168.57.102
Processes: 99
```

```
charith@bastion:~$ pwd
/home/users/charith
```

3. Bastion Auth Log

```
Mar 8 18:10:15 puppetmaster sshd[10671]: Accepted publickey for charith from
192.168.57.1 port 64349 ssh2: RSA 6f:5e:c9:a6:8f:47:3f:02:b8:c6:fa:fe:08:df:10:c5
Mar 8 18:10:15 puppetmaster sshd[10671]: pam_unix(sshd:session): session opened
for user charith by (uid=0)
```

Appendix X: Testing server behavior when LDAP server is down

System output when the SSH server is not able to connect to the LDAP, hence the key based authentication fails and goes to the next authentication mode; password based authentication

```
dilan@Dilans-MacBook-Pro:~$ ssh -v charith@192.168.57.102 -i /tmp/id_rsa_charith
OpenSSH_6.2p2, OpenSSL 0.9.8r 8 Dec 2011
debug1: /etc/ssh_config line 20: Applying options for *
debug1: /etc/ssh_config line 102: Applying options for *
debug1: Connecting to 192.168.57.102 [192.168.57.102] port 22.
debug1: Connection established.
debug1: identity file /tmp/id_rsa_charith type 1
debug1: identity file /tmp/id_rsa_charith-cert type -1
debug1: Server host key: RSA a1:53:b1:71:f4:dc:50:51:ed:36:6e:01:ea:93:c3:8c
debug1: Host '192.168.57.102' is known and matches the RSA host key.
debug1: Found key in /Users/dilan/.ssh/known_hosts:72
debug1: ssh_rsa_verify: signature correct
debug1: SSH2_MSG_NEWKEYS sent
debug1: expecting SSH2_MSG_NEWKEYS
debug1: SSH2_MSG_NEWKEYS received
debug1: Roaming not allowed by server
debug1: SSH2_MSG_SERVICE_REQUEST sent
debug1: SSH2_MSG_SERVICE_ACCEPT received
debug1: Authentications that can continue: publickey,password
debug1: Next authentication method: publickey
debug1: Offering RSA public key: /tmp/id_rsa_anura
debug1: Authentications that can continue: publickey,password
debug1: Offering RSA public key: /tmp/id_rsa_charith
debug1: Authentications that can continue: publickey,password
debug1: Next authentication method: password
charith@192.168.57.102's password:
```


References

- [1] OpenSSH Features (no date) [Online]. Available : <https://www.openssh.com/features.html>
- [2] Tore Torsteinbo. (2011, November). Remote Login. [Online]. Available: http://www.dietmueller.at/download/07_RemoteLogin.pdf
- [3] T. Ylonen and Lonvick. (2006, January). SSH Authentication Protocol.[Online]. Available: <https://tools.ietf.org/html/rfc4252.html>
- [4] Rajdeep Bhanot and Rahul Hans. (no date). A Review and Comparative Analysis of Various Encryption Algorithms(Vol. 9, No. 4 (2015)). [Online]. Available : http://www.sersc.org/journals/IJSIA/vol9_no4_2015/27.pdf
- [5] K.Aman and Dr. J.Sudesh and M.Sunil. (2012, July, 7). Vol 2,Issue 7.[Online]. Available: http://www.ijarcsse.com/docs/papers/July2012/Volume_2_issue_7/V2I700262.pdf
- [6] Ankit Gambhir. (2014, April). Vol 3,No4. RSA Algorithm or DES Algorithm. [Online].Available: <http://borjournals.com/a/index.php/jecas/article/viewFile/1643/1038>
- [7] SSH: The Secure Shell The Definitive Guide. [Online]. Available : http://docstore.mik.ua/oreilly/networking_2ndEd/ssh/ch02_04.htm
- [8] Dario Berzano.(2012,November,30) SSH AUTHENTICATION USING GRID CREDENTIALS. [Online]. Available: http://web.infn.it/CCR/images/stories/upload_file/note_ccr/infn-12-20_42.pdf
- [9] Bill Bryant.(1988, February). Theodore Ts'o.(1997,February). Designing an Authentication System: a Dialogue in Four Scenes. Copyright 1988, 1997 Massachusetts Institute of Technology. [Online]. Available: <https://web.mit.edu/kerberos/dialogue.html>
- [10] Don Jones. (Oct 11, 2012). MicroNugget: How Does Kerberos Work?. [Online]. Available: <https://www.youtube.com/watch?v=kp5d8Yv3-0c>
- [11] B. Clifford Neuman and Theodore Ts'o . The Kerberos Network Authentication Service.(1994, September) Volume 32, Number 9, pages 33-38.[Online]. Available: <http://gost.isi.edu/publications/kerberos-neuman-tso.html>
- [12] OpenLDAP Foundation. Introduction to OpenLDAP Directory Services. [Online]. Available: <http://www.openldap.org/doc/admin24/intro.html>
- [13] Hynek Schlawack. LDAP: A Gentle Introduction. (13 February 2007). [Online]. Available: <https://hynek.me/articles/ldap-a-gentle-introduction/>
- [14] EMC Corporation. RSA SECURID® Hardware Tokens. [Online]. Available: <https://www.rsa.com/content/dam/rsa/PDF/h13821-ds-rsa-secrid-hardware-tokens.pdf>
- [15] RSA SecurID Software Token,Google Authenticator. [Online]. Available: <https://play.google.com/store/apps/details?id=com.rsa.securidapp&hl=en>, <https://play.google.com/store/apps/details?id=com.google.android.apps.authenticator2&hl=en>
- [16] Michael Holley. (September 29, 2015). How To Set Up Multi-Factor Authentication for SSH on Ubuntu 14.04. [Online]. Available: <https://www.digitalocean.com/community/tutorials/how-to-set-up-multi-factor-authentication-for-ssh-on-ubuntu-14-04>
- [17] Kurt Dillard. Collective Technologies, Inc.[Online]. Available : <https://www.sans.org/security-resources/idfaq/what-is-a-bastion-host/2/11>

- [18] Stuart Scott.(2015, December,1). AWS Security: Bastion Host, NAT instances and VPC Peering [Online]. Available: <http://cloudacademy.com/blog/aws-bastion-host-nat-instances-vpc-peering-security/>
- [19] Mike Pope .(21 MAY 2014). Best Practices, Enterprise, How-To Guides, Networking. [Online].Available: <https://aws.amazon.com/blogs/security/securely-connect-to-linux-instances-running-in-a-private-amazon-vpc/>
- [20] Google Security Whitepaper. (2017, January, 10). [Online]. Available: <https://cloud.google.com/security/whitepaper>
- [21] sshd_config(5) - Linux man page. [Online]. Available: https://linux.die.net/man/5/sshd_config
- [22] Sun Microsystems, Inc. (2017-05-03). NSSWITCH.CONF(5) Linux Programmer's Manual. [Online]. Available: <http://man7.org/linux/man-pages/man5/nsswitch.conf.5.html>
- [23] Shichro An. (2015, April, 17). Setting up OpenLDAP server with OpenSSH-LPK on Ubuntu 14.04. Copyright 2015, Shichao An. [Online]. Available: https://blog.shichao.io/2015/04/17/setup_openldap_server_with_openssh_lpk_on_ubuntu.html
- [24] New Zealand Government ICT Functional Leader, the Government Chief Information Officer. (09/05/2014). Password Vulnerabilities and Attacks. [Online]. Available: <https://www.ict.govt.nz/guidance-and-resources/standards-compliance/authentication-standards/password-standard/5-password-vulnerabilities-and-attacks/>
- [25] Chwei-Shyong Tsai, Cheng-Chi Lee, Min-Shiang Hwang.(Sept. 2006). Vol.3, No.2, PP.101–11. Password Authentication Schemes: Current Status and Key Issues. [Online]. Available: <http://ijns.femto.com.tw/contents/ijns-v3-n2/ijns-2006-v3-n2-p101-115.pdf>
- [26] Dan Boneh, Twenty Years of Attacks on the RSA Cryptosystem [Online]. Available: <http://crypto.stanford.edu/~dabo/papers/RSA-survey.pdf>
- [27] RFC 4519. (June 2006). LDAP: Schema for User Applications.[Online]. Available: <http://www.rfc-editor.org/rfc/rfc4519.txt>
- [28] OpenLDAP Foundation. Copyright 2011. Security Considerations. [Online]. Available: <https://www.openldap.org/doc/admin24/security.html>
- [29] sshpass(1) - Linux man page. [Online]. Available: <https://linux.die.net/man/1/sshpass>