Cloud Security: Inter-Host Docker Container Communication using Vault Dynamic Secrets

Ramesh K V, G T Raju

Abstract*: In this paper we attempt to address Inter-Host Docker container communications security issues by incorporating a latest approach provided by Vault Hashicorp dynamic secret mechanism for managing SSH keys and server credentials. A simulation environment is prepared for Inter-Host container communication consisting of one host running locally and the peer host running as an AWS EC2 instance in cloud. Industry standard monitoring tool Grafana is used in the simulation environment to highlight the security impacts for any organization. We also draw special attention to some of the security vulnerabilities in docker container like ARP spoofing, Integrity of the docker host and containers and MAC flooding attacks. We try to list some best practices to be followed when using docker containers in any production deployments.

***Keywords*: Docker containers, Dynamic secrets, Grafana, Cloud Security, Vault Hashicorp**

I. INTRODUCTION

Docker [2] is an open-source system of software containers. It is an isolated portion of the host computer, sharing the host kernel (OS) and even its binaries/libraries if appropriate. Containers isolate an application and its dependencies into a self-contained unit that can run anywhere. Containerization is a powerful way to package and deploy application in cloud. Docker is a deployment tool that packages your code along with all the dependencies into an application container in such a way that it doesn't require a full-fledged Virtual Machine to run. Container provides operating-system-level virtualization by abstracting the user space. Docker containers isolate applications from one another and from the underlying infrastructure.

.

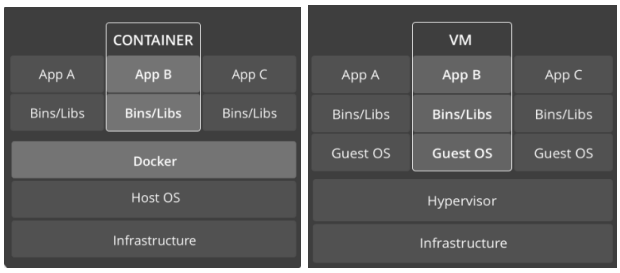

Fig. 1. Architecture of Container and Virtual Machine.

Revised Manuscript Received on December 15, 2019.

Mr. Ramesh K V, PhD Scholar & Software Consultant at Tanmay Consultancy Pvt Ltd Wokingham UK Email: kvramesh@gmail.com

Dr. G T Raju, Vice-Principal, Professor & HOD RNSIT Bangalore India Email: gtraju1990@yahoo.com

As per NIST definition [6], Cloud Computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources. In recent years Cloud computing has emerged the go-to method for most organizations across the globe to deploy managed services. Managing in cloud means distribute the local workload in cloud to make the applications and services more reliable, more flexible, portable, scalable on demand and most importantly capital-expenditure free. There are various cloud providers like Azure, Google & Amazon which provides most services for customized revenue model based on the size of the organization.

Docker containers share the host operating system kernel. The startup time is comparatively very quick usually in terms of seconds when compared to minutes of VM startup time. Containers use less compute and RAM. Docker Images are constructed from file system layers and share common files. This minimizes disk usage and as the image size is smaller, image downloads are much faster. Docker containers are based on open standards and run on all major Linux distributions, Microsoft Windows, and on any infrastructure including VMs, bare-metal and in the Cloud. Docker containers isolate applications from one another and from the underlying infrastructure. Docker provides the strongest default isolation to limit app issues to a single container instead of the entire machine. Docker containers are lighter than Virtual machines, and we can run a ton of processes on a reasonable sized host where Deploying and scaling is relatively is easy [7].

We can categorize the analysis of docker container security in to following broad areas:

- Intra-Docker container communication vulnerabilities where one container having adverse impact on another container within the same host.
- Docker container to host vulnerabilities where when malicious application within the container could gain root access in to host kernel there by compromising the entire system.
- Inter-Host Docker container vulnerabilities where containers are running in different hosts. Most of the latest application deployment are done using micro service architecture which will fall in to this category.

In this paper we are concentrating on Inter-Host Docker container vulnerabilities.

II. DOCKER SECURITY CONSIDERATIONS

The basic architecture of Virtual Machines (VM) on any host will have its own kernel and communicates with the Host OS via hypervisor layer. Applications running on VM thus have two layers of isolation in order to breach any restrictions and attack Host kernel. As the Docker containers directly can use host kernel, makes it easier for any malicious containers to attack host kernel and the consequences can be very dangerous. Once the compromised containers gain access to host kernel could lead to attacks like denial of service (DoS attack) and privilege escalations [9].

Integrity of Docker Host Issue: It important to protect the Host server running the container from external threats. In order to handle this aspect, AWS provides network security groups. We can create several security groups and associate each EC2 instance to a specific group. Each security group will have specific outbound and inbound rules with respect to what protocols and ports are allowed to access in each direction. In our simulation example we have opened the ports 3000 and 8086 for Grafana and influx database so that these can be accessed externally. All other ports are closed by default. Sufficient prominence should be given to container auditability, docker management and authentication of the docker host.

Integrity of Docker containers: This could be addressed using taking enough care on properly validating the container contents by implementing manifest for each container. The Manifest can guarantee the contents are not tempered during transit and also validates the source of the components within container such as libraries and executables.

ARP Spoofing and MAC Flooding Attacks: The docker host network bridge interface is vulnerable for Address Resolution Protocol (ARP) spoofing. This is when an attacker tries to associate attackers MAC address with the IP address of a trusted host. This type of ARP spoofing can intrude in to containers too as container interface directly connected with host network bridge interface. This can further lead to media access control (MAC) flooding attacks where attacker sends many ethernet frames each containing different source MAC addresses.

Some of the best practices when using Docker containers are:

1. Using trusted repositories for container images as these repositories would have scanned for any vulnerabilities.
2. Know what is in your Container for any runtime scripts which might download malicious code during container execution. Make sure everything that a container runs must be declared and included in the static container image.
3. Set resource limits for each container so that it is easy to monitor for containers competing for host resources. These controls are done using orchestration frameworks such as Kubernetes or Docker Swarm.
4. Avoid running containers in super-privileged mode. Grant these privileges to protected resources.
5. Emphasis on Container lifecycle management which involves creating, updating and deletion of Containers.

III. CONTAINERS AND MICROSERVICES

Microservice is a mechanism of building distributed units of software components which predominantly has single function with well-defined interfaces and operations to the external modules. For example, separating database from the UI component makes it lot easier to manage both the services. This also makes it easier for scaling based on the demand. Companies like Netflix, eBay, Amazon, Twitter, PayPal all evolved from monolithic architecture to microservice architecture [3]. Monolith application is built as a single, autonomous unit. Any modification to application will require full upgrade of all the deployment. On the contrary in Microservices, if database is modified only database microservices needs deployed without touching UE services. In monolithic architecture scaling becomes very complex and expensive as you would need full application scaled. [12].

Microservices also give us flexibility to be implemented in different programming languages and makes it easier to follow Agile software method in delivering a fully tested service. Each microservice also has flexibility to select different data storage techniques like influx, PostgreSQL, MongoDB etc. In the simplest form, Microservices help build an application as a suite of small services, each running in its own process and are independently deployable.

In most modern-day deployments, Microservices run in docker containers. As containers instantiate very quickly when compared to legacy Virtual Machine, this adds great value for customer experience. Containerization is OS level virtualization which isolates applications from each other and the underlying infrastructure. Furthermore, containers run on any computer on any infrastructure and in any cloud.

IV. SIMULATION

Simulation consists of a local host server running Centos 7 OS acting as a Docker Host system. On this server we have built docker containers each running a microservice. Microservices generate specific event logs which are directly sent over to an influx database which is running on a container in AWS cloud EC2 instance. We have made use of monitoring tool Grafana through which we can generate useful graphs based on the contents of the influx database. We have also installed Telegraf software which can generate system level stats directly fed in to influx database. Telegraf is installed on the Docker Host to monitor the overall system health like CPU, memory, idle time etc.

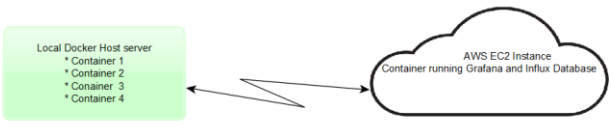

Fig. 2. Inter-Host container communication architecture.

Some of the Grafana snapshots taken when the containers are generating event logs sent over to AWS EC2 instance. We can clearly see the direct relation with respect to CPU utilization and application event log stats.

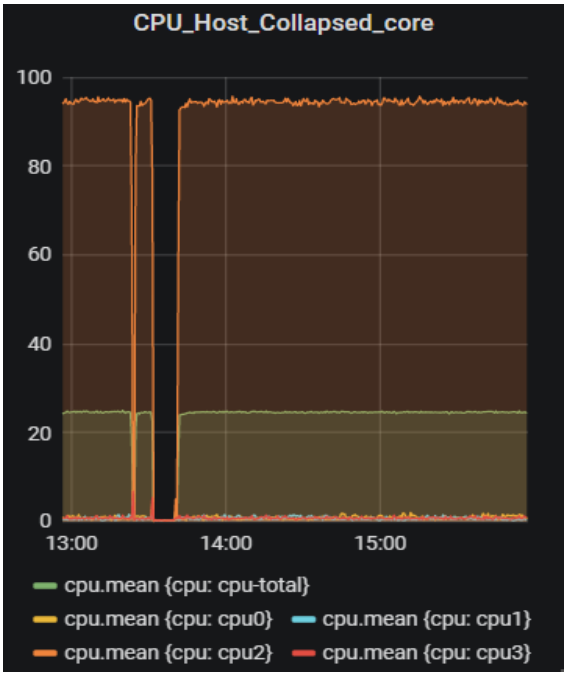

Fig. 3. CPU usage of host displayed on AWS EC2.

The above graph is based on telegraf stats from the host shows there is CPU idle time between 13-14:00. This could be the results of a system outage on the host machine due to various reasons like software crash, planned system maintenance due to software upgrade etc.

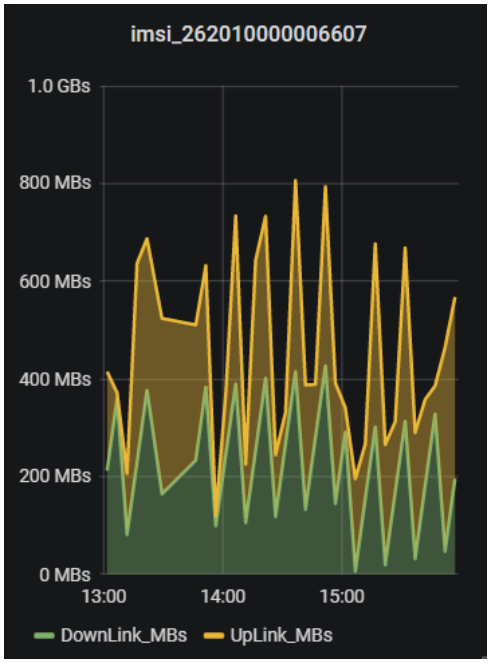

Fig. 4. Data Rate of a mobile on Grafana AWS EC2.

The above graph is based on microservice event log stats from one of the containers which shows that microservice momentarily restarting. In a typical production deployment these monitoring graphs play an important role analyzing any system outages directly impacting company revenues.

V. CLOUD INSTANCE VULNERABILITIES

Migrating enterprise applications to the cloud has many benefits but also carries drawbacks especially where architectures and applications have not been designed as 'cloud-native' from the beginning. The primary security goal of any deployment will be to deliver trusted secured communications and login capabilities. However, this is made substantially harder in the cloud environment as the compute nodes are often dynamic instances that are orchestrated or scaled without human intervention and are not tied to static IP addresses.

When managing applications, it is common to configure servers with an SSH key for host access or provide software with server and client SSL certificates. Both these secret types are used to ensure secured communications either between an administrative user and the server (SSH) or between applications (SSL).

Recent survey shows that 60-70% organization store the server credentials either in plain text or in some common database like git hub. In general, is difficult to prevent the unauthorized distribution of secrets, impossible to control and audit use of these secrets and finally rotation of credentials in the event of breach is very hard with long lived and manually deployed or hardcoded keys. All these problems become worse in the dynamic cloud environment and this type of vulnerability is termed secret sprawl.

Ideally systems would be deployed with a solution that can:

- centralize the management and issuing of different types of secrets
- provide role-based limitations on which secrets different clients can request
- authenticate requesting clients
- generate audit history for all activity
- providing a single abstracted API to different types of key management system
- ensure that all secret information is securely encrypted in storage.
- be fully automated so that secret lifetime can be kept very short removing the requirement for revocation lists as any lost certificate or barred user or service would forbidden access very rapidly.

VI. VAULT HASHICORP

The Vault application from Hashicorp is designed to deliver a solution that meets all these requirements and is specifically tailored to the needs of a cloud deployment.

It provides an arbitrary secret storage service in the form of a key value store. Different storage layers can be configured and never see the unencrypted data. The abstract interface is presented as a virtual filesystem that can be mapped onto many different secret engines including Hardware Security Modules (HSM), Cloud infrastructure providers (AWS, GCP, Azure), PKI (SSL certificate) and SSH services.

As a cloud native service, it supports clustered high availability and is very capable of generating short lived secrets.

A. System Architecture

Vault architecture contains following components.

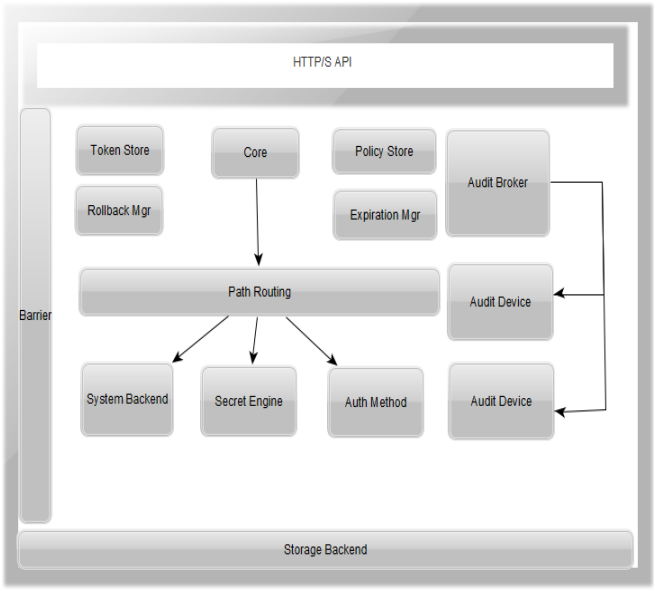

Fig. 5. System Architecture of Vault

- Storage Backend is responsible for storing the encrypted data and configured during startup.
- Barrier is the region where encrypted data is accessed and stores only the verified data while it is storing. All the communication between application API and storage backend must pass through this Barrier.
- Secret Engine is responsible for managing secretes. In this paper we are considering only SSH secretes but there are other secretes like MySQL, AWS, Google Cloud, Consul etc which is for further studies.
- Auth Method is the way in which users are authenticated. Once authenticated these users can access appropriate policies. There are various authentication method like userpass, gitbub, okta, TLS Certificates, LDAP etc. In this paper we are using userpass method.
- Policy store contains ACL rules. This is where admins can create specific policy rules based on the user category.
- Audit Device is responsible for managing audit logs.
- Token store is used to verify the identity of clients and thus apply specific Policy rules.

B. Dynamic secretes

As discussed earlier long-lived secrets are very hard to manage and contain and there are many examples of applications or users accidentally (or deliberately) sharing or leaking this information.

Once a secret has been compromised it is necessary to revoke the previous access rights and deploy new credentials to all affected services and users. Short lived secrets are much preferred as they are much less sensitive due to their limited life and scope of access. It is possible to take this concept further and create fully dynamic secrets. In this case a secret is created on demand and is unique to a client. The secret has a short lifetime and the client will need to request a new secret regularly in order to maintain access to the service.

Vault is capable of delivering this type of dynamic secret as a service on top of many different services including many that have no concept of this type of operation themselves. For example, database login credentials can be managed by Vault such that all users and applications are provided with short lived access passwords that can be rotated with ease.

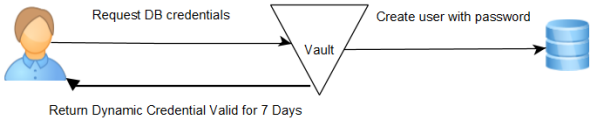

Fig. 6. General DB access process using Vault

C. One Time Password

The most extreme version of a short-lifetime dynamic secret is a one time password. In this case a client specific unique access token that is only valid for a single use is generated on request [1].

This can be used to provide a single use SSH access key for logging into a server.

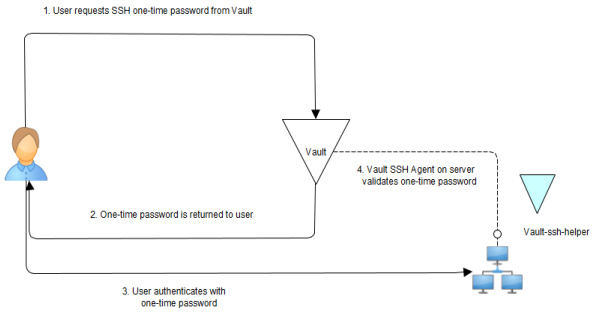

Fig. 7. OTP process using Vault

The Vault service integrates with the SSHd service on the host and is able to provide one time passwords so that the primary access credentials to the server are never exposed. Although there is an increase in the security surface relating to the interaction between the Vault service and the hosts, this can be managed carefully and dramatically offsets the ease of management of secure SSH access to dynamically changing cloud computing resources.

D. Signed SSH Certificates

The signed SSH certificates is platform independent and easy to setup the environment compared to OTP. It uses Vault's powerful CA capabilities and functionality built into OpenSSH, clients can SSH into target remote hosts on cloud using their own local SSH keys. In this method we use Vault as a certificate authority for signing SSH keys as mentioned int the below flow diagram. Vault validates the requests from the user for access to the remote hosts instance.

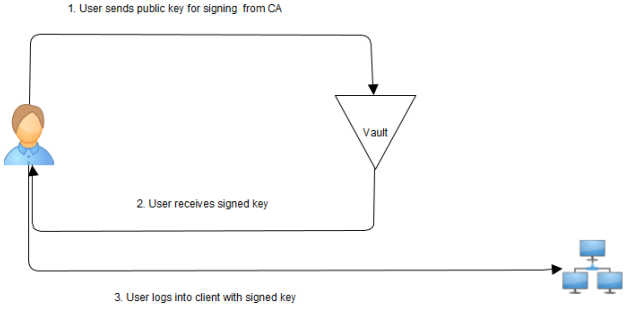

Fig. 8. Signed SSH certificates using Vault

E. mTLS

Vault also supports a PKI secret engine for generating X.509 certificates that can be used for SSL/TLS. This can support the use of fully mutual TLS encrypted communications between all cloud services where the lifetime of the certificates can be kept very small (e.g. 72h). This is possible because the services are able to request certificates programmatically avoiding the manual process of generating a private key and Certificate Signing Request (CSR) which would then need to signed by a Certificate Authority (CA).

Vault has built in authentication and authorization capabilities that mean this process can be fully automated once it has been configured to use either a self-signed CA or more commonly an intermediate CA provided by the user.

VII. RESULTS AND DISCUSSION

In this paper we have implemented OTP and Signed SSH Certificates modes which are part of Vault SSH secrets engine.

These two modes provide secure way of authentication and authorization via SSH protocol to access remote host machines. Vault servers are configured using a file which is in either HashiCorp configuration Language (HCL) or JSON format.

A. One Time Password

In OTP mode, Vault server will issue a One-Time Password each time the authenticated client requests SSH access to a remote host. Vault ssh helper command plays an important role of client verification. The OTP used by the client will be sent to ssh helper command which will do the validation of OTP with the Vault server. Once validation is through, Vault server deletes the OTP making sure this OTP can be used only once by the client. Next attempt client will be provided with a different OTP. This way we leverage dynamic secrete. Every login attempt and the corresponding TTL information will be validated and recorded at the audit logs which can be very handy during future references.

OTP method requires vault ssh helper binary to be installed and SSH configuration changed to enable keyboard-interactive and redirect its client authentication responsibility to vault-ssh-helper. Vault-authenticated users contact the Vault server and retrieve an OTP issued for a specific username and IP address. While establishing an SSH remote connection to the host, the vault ssh helper binary reads the OTP from the password prompt and sends it to the Vault server for verification. Client authentication is successful (and the SSH connection allowed) only if the Vault server verifies the OTP. Once the OTP has been used a single time for authentication, it is removed from Vault and cannot be used again [1].

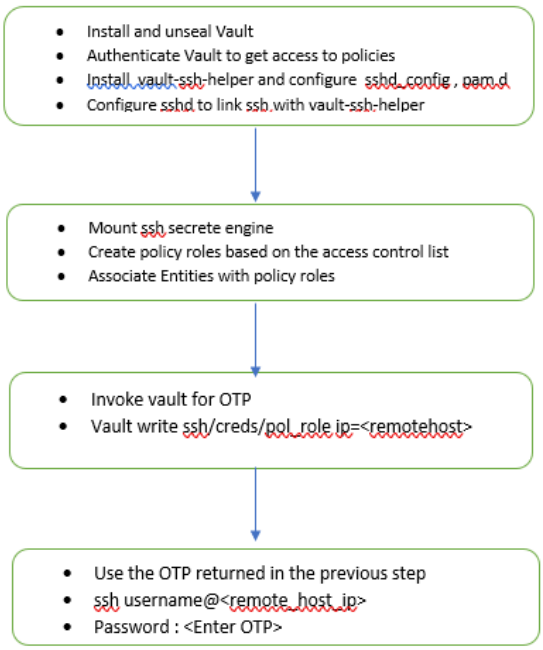

Fig. 9. Flow diagram for the OTP implementation:

Example output of the OTP key request and remote host connection using OTP key

```
$ vault write ssh/creds/otp_key_role ip=192.0.2.10
Key                Value
---                -----
lease_id           ssh/creds/otp_key_role/234bb081-d22e-3762-3ae5-744110ea4d0a
lease_duration     768h
lease_renewable    false
ip                 192.0.2.10
key                f1cb47ad-6255-0be8-6bd8-5c4b3b01c8df
key_type           otp
port               22
username           ubuntu
```

```
$ ssh ubuntu@192.0.2.10
Password: <Enter OTP>
```

Please note that if we just issue another request then we get a different OTP key as the name suggests. This is the main advantage of this method.

B. Signed SSH Certificates

This method leverages the powerful CA capabilities and functionality which is built in to OpenSSH. Client can request their SSH key be signed, the Vault SSH secretes engine must be configured for this mode. This mode is more powerful yet simplest to configure when compared to other Secretes Engines from Vault. The steps to implement this method is briefed in the flow diagram below. The key highlight of this is we can configure the time-to-live (TTL) of the signed key can be configured as low as 30 seconds and it can prove very powerful in certain circumstances.

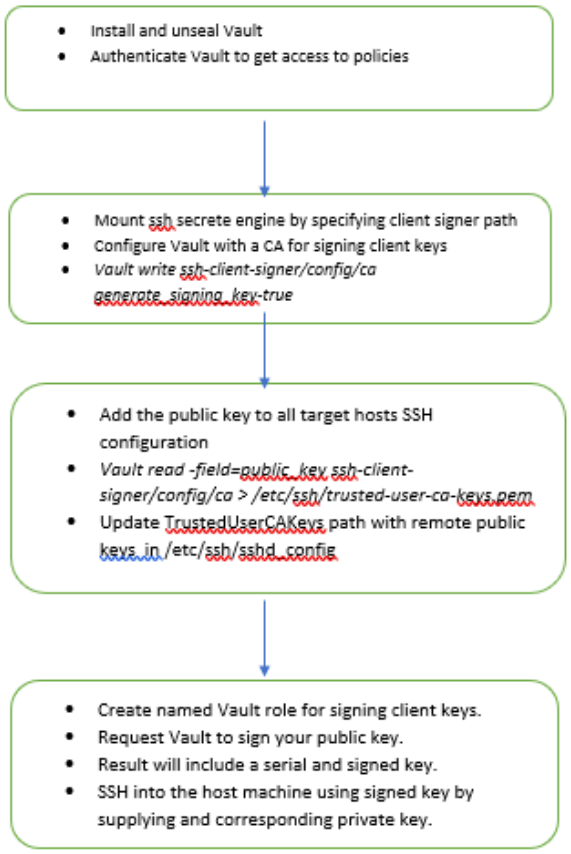

Fig. 10. Implementation flow chart for Signed certificates

Example output of the getting the signed certificate, saving the received signed certificate and invoking remote host machine:

```
$ vault write -field=signed_key ssh-client-signer/sign/my-role \
    public_key=@$HOME/.ssh/id_rsa.pub > signed-cert.pub
```

```
$ ssh -i signed-cert.pub -i ~/.ssh/id_rsa username@10.0.23.5
```

Applying the two methods of Vault, we have seen the advantages of securing the credentials in one safe place and retrieving dynamic secretes to access the remote host was very efficient. As a result of using Vault, we don't have to worry about credentials in the hands of employees leaving the organization which could potentially lead to unethical use of credentials. These methods eliminated the main issue of secret sprawl and unauthorized distribution of the credentials. Enabling the Audit logs at Vault server we saw critical information about any security breach. The audit logs contain the full request and response objects for every user interaction with Vault server. If there is a rogue user trying to get credential from Vault, this information will be logged too. With the help of Audit logs, we can trac which credentials were accessed by whom and when very easily.

From our test results we have seen how a authenticated user having access to correct Policy rules can requests for OTP tokens and login to remote AWS instances. As a negative test we also tried to create a dummy user and associated default policy which does not allow request for OTP keys, we got appropriate error from Vault. We could also cross verify the same in the audit logs that clearly pointing the user details, time of erroneous access to OTP secrets.

There are certain drawbacks of the two methods we just highlighted. With SSH secrets engine type, remote host machine could connect back to Vault server with some malicious intensions. This could potentially lead to dangerous consequences. If compromised, an attacker could spoof the Vault server returning successful request and take unwanted advantages of the spoofing. This type of attacks can be mitigated by using mutual Transport Layer Security (mTLS). This will be the area of interest for the future research.

VIII. CONCLUSION

The aspect of Inter-host container communication and its security considerations are going to be very crucial in coming years as more and more software companies are making shift towards Microservices deployment model. In order to address the security challenges in Inter-Container communication, we have tried to explore two capabilities of Vault in this paper. We have also listed the positive consequences of using Vault Dynamic secrets. There are numerous other capabilities like mTLS, Vault on platforms like Kubernetes, & various other secrets engines can be further explored.

ACKNOWLEDGMENT

We thank Mr. Riki Dolby VP Engineering Infocom UK for his guidance on Vault HashiCorp applications. His continuous directions and feedback have helped preparing this paper. We also would like to thank Dr. Keshava Munegowda consultant Engineer at DellEMC Bangalore India, for constant support and review of the paper work.

REFERENCES

1. Literature available at https://www.vaultproject.io/docs/install/index.html , 2019
2. Literature available at URL:https://www.docker.com/what-container , 2019
3. Literature available at https://www.nubera.eu/containers-microservices-nubera , 2019
4. Literature available at URL:https://aws.amazon.com/what-is-cloud-computing/ , 2019
5. Literature available at URL:https://ieeexplore.ieee.org/document/7557545/ Securing Docker Containers from Denial of Service (DoS) Attacks, 2016
6. P. Mell and T. Grance, "The NIST Definition of Cloud Computing - Recommendations of the National Institute of Standards and Technology," in NIST Special Publication 800-145, Gaithersburg, MD 20899–8930, 2011
7. https://ieeexplore.ieee.org/document/7796159/ Securing Cloud Containers Using Quantum Networking Channels
8. https://ieeexplore.ieee.org/document/7530217/ Docker container security via heuristics-based multilateral security-conceptual and pragmatic study
9. Dirk Merkel,"Docker: lightweight Linux containers for consistent development and deployment", Linux Journal, Vol. 2014, No. 239
10. Literature available at https://www.vaultproject.io/docs/internals/architecture.html ,2019
11. Literature available at https://abridge2devnull.com/posts/2018/05/leveraging-hashicorp-vaults-ssh-secrets-engine/ , 2019
12. Literature available at https://smartbear.com/solutions/microservices/ , 2019

AUTHORS PROFILE

Mr. Ramesh K V, currently working as software consultant at Tanmay Consultancy Ltd based in the UK. He is also PhD scholar in VTU university India. He has over 15 years of Industry experience working with companies like Ruckuswireless - previously known as IntelliNet Technologies, Airspan Networks, Lloyds Bank & Quortus Networks.
He has close to 3 years of teaching experience in KIT Tiptur. His area of interest includes Cloud Computing, Telecommunications and Network security. He has MTech CSE from BMSCE, Bangalore

Dr G T Raju, currently working as Vice Principal, professor & head at CSE Dept, RNSIT. He has 25 years if teaching & 14+ years of research experience. He has published over 60 papers in International journals, conferences. He organized 24 workshops/FDP/conferences & delivered 24 invited talks. He has received over 25 lakhs research grants from AICTE and VTU. He is life member of ISTE & CSI. His research interests are pattern recognitions, web mining, image processing & information retrieval. He has held Various positions at VTU University.