Using Software Containers for Privileged Access Management in Cloud Environments: A Novel Approach to Handle Access Management for Cloud-based Networks

Marleen Steinhoff

Munich University of Applied Sciences, Munich, Germany
E-mail: marleen.steinhoff@hm.edu

Received 15 July 2020; Accepted 01 September 2020;
Publication 30 January 2021

Abstract

This paper presents a novel approach for privileged access and session management using containers. Current solutions are built using proxies, proxy suites or jump servers, but they do not cater for third party remote access security requirements, have additional vulnerabilities and have scalabilty limitations.

The novelty of the solution proposed in this paper is a global orchestrator that instantiates a purpose-built container adapted to the virtual network functions' system. Every container has a logging function, a pre-defined time-to-live and one-time-credentials. This approach is secure because the containers isolate different connections, privileges are restricted, permissions are always time-limited and the provider has full control over the sessions. The solution brings several other security enhancements, discussed in this paper.

Keywords: Cloud security, container, privileged access management, cloud computing, third-party access.

Journal of NBICT, Vol. 1, 297–310.
doi: 10.13052/nbjict1902-097X.2020.013

1 Introduction

Cloud networks built with third-party solutions need to provide access to the network functions by several parties for operations and management. This access is necessary to perform all required actions in the software life cycle, including deployment, operation, and maintenance [1]. However, this access, known as privileged access management (PAM), can enable internal and external attackers to carry out attacks.

As shown in [2], one of the main problems with PAM in cloud environments is credential misuse. This includes external attackers as well as attacks from malicious insiders. Over two-thirds of attacks are carried out using stolen credentials [3]. But not only stolen credentials can be used to carry out attacks. So-called malicious insiders already have privileged access to the network or network functions which they can misuse and are therefore seen as the biggest security threat in public clouds by 30% of cyber security experts [4].

Referring to [2], another problem with PAM is the heterogeneity in cloud networks, that comes in three aspects:

First, heterogeneity in different network functions (NFs) provided by solution providers that are built for different purposes and therefore support different protocols. Second, heterogeneity in different user groups including third parties requesting access to the NFs. As the users connect to NFs remotely, the cloud provider doesn't know who connects to his network. Third, heterogeneity in different software layers a cloud network comes in. The layers all have to be accessible by users and maintainable by providers.

Solutions available to the cloud providers to given problems are still facing attacks thus proving that they are not sufficiently secure. Therefore, current solutions do not provide enough security. A closer look at the leading software solutions reveals that they all use similar approach: Proxies, proxy suites or jump servers are used to establish connections between users and NFs. This approach leads to a single point of failure making it a valuable target for attackers. Therefore, a novel approach is needed which solves the current problems, which is presented in this paper.

The leading software solutions for PAM are analyzed in Section 2 and their limitations discussed in Section 3. The limitations derived requirements are in Section 4. Section 5 shows the novel solution using containers. This solution is discussed in Section 6. The Paper comes to its end with Section 7 about future work and the conclusion in Section 8.

2 Existing Solutions

Numerous solutions have been developed for PAM for cloud-native networks. Six leading software solutions will be analyzed in the following subsections regarding their implementation architecture.

CyberArk

CyberArk offers the Privileged Access Security suite together with Alero for access through a web portal. The solution uses proxy or jump server mechanisms [5, 6].

BeyondTrust

BeyondTrust's Privileged Remote Access (powered by Bomgar) offers a suite with a jump server approach. Their solution includes different software components used for their PAM solution [5, 7].

Centrify

Centrify's Privileged Access Service uses a jump server, so-called jump box [8]. According to Gartner [5], these solutions can handle more connections than other solutions but are still limited in the number of requests the solution can handle.

PrivX

PrivX from SSH Communications Security provides a solution using scalable microservices for PAM [9]. Looking at the documentation, PrivX uses a suite of microservices, containers and proxies [10].

PingAccess

PingAccess from Ping Identity provides access to applications and APIs for authorized users. To achieve that, they use a proxy called PingAccess proxy [11].

CA Technologies by Broadcom

The CA privileged access manager from CA Technology offers a suite of PAM products and supports session management flexibly through jump server and proxy mechanisms [5].

Gartner analyzed 14 different PAM and PASM solutions in their 2018 Magic Quadrant for Privileged Access Management including the leading software solutions and also niche players [5]. All of the solutions use a similar architecture design: Either a proxy, a proxy suite or a jump server is used to establish the connections and to grant privileges.

3 Limitations of Existing Solutions

In the introduction, six problems have been mentioned: External attackers carrying out attacks (P1), attacks from malicious insiders (P2), heterogeneity of provided network functions regarding supported protocols (P3), unknown users connecting to network functions remotely (P4), access to different software layers within the cloud network (P5) and the proxy solutions creating a single point of failure (P6). But there are more limitations the current solutions come with.

As shown in the previous section, these leading solutions use either proxies, proxy suites or jump servers.

A proxy is a software on a server that enables the server to act as an intermediary, processing requests from clients without connecting them directly. Proxy suites consist of a number of proxies to increase security. As a specialized form of proxy, a jump server is a special type of intermediate server handling access from outside the network to systems behind firewalls. This access allows administrators to run the systems inside the network without exposing them.

These technologies are being used to handle access to cloud systems. Even though PrivX uses microservices and containers in their solution, the connection is still launched via a single proxy [10].

Using a proxy approach limits the number of possible connections (P7). Referring to Gartner [5], the CA Privileged Access Manager from CA Technology can handle more simultaneous connections than any other product they evaluated. However, the amount of connections is still limited by the architecture itself.

Another problem with bundling all connections through a single server is that an attacker can directly access any network element as soon as the attacker has access to the connection device (P8). Therefore, proxies or jump servers are a high-value target for attackers.

Even when PAM solutions are implemented in cloud networks, workarounds are existing, next to the PAM solution. For example local

credentials (P9) or weak fall-back solutions (P10) [2]. Reasons for existing workarounds are changing requirements due to change in demand. If the implemented solution is not flexible enough (P11) to be adapted to the new requirements in time, workarounds are implemented as cloud providers have already invested in PAM solutions. These workarounds can create vulnerabilities and also lead to incomplete logging while log file integrity is mandatory to detect attacks on a network.

4 Requirements

The requirements for the solutions are derived from the problems (P1–P11) mentioned in the previous section and marked using an "R" followed a number (R1–R15).

To prevent external to carry out attacks (P1), a secure PAM solution needs to provide distinct connections between the user and NF (R1). The connection shall grant only least privileges (R2) to avoid users from performing unauthorized actions. But even if a user is authenticated and the access privileged, every action shall be logged to allow tracing back actions and detect attacks (R3).

The logging backend should be tamper-proof and can be combined with forensic tools to manage and analyze the logs (R4). In cloud environments, Multi-Factor-Authentication (MFA) is highly recommended against credential misuse by external attackers [12–15] and shall be implemented to authenticate the users (R5). Since credentials can be stolen by attackers, the credentials should be regenerated for each connection (R6).

These requirements also help to prevent malicious insiders from carrying out attacks (P2). To make these measures effective, connections shall only be possible through logged containers.

Even if a user is authenticated, every action shall be logged to allow tracing back actions and detect attacks.

Nobody should be able to connect to NFs directly (R7). This includes, that workarounds or local credentials (P9) shall not be implemented on the NFs (no workarounds R8, no local credentials R9).

Developing a secure solution that takes the above requirements into account allows the cloud provider to grant access to users from different user groups (P4). Although the cloud provider does not know who is actually using the provided access to the cloud network, this method ensures that rights are limited to what is necessary and all actions are logged. The solution should

be reused to grant access to NFs on every software layer and in every network segment (P5, R10). This measure makes it easier to maintain and harden PAM for the cloud network with a single solution.

To handle the numerous protocols of the NFs a flexible, modular solution is needed that can be modified or extended if needed. The solution should be easily extended by further protocols and configurations (P11, R11).

In order to avoid implementing a main instance that becomes a single point of failure, the main instance has to be well-protected and requires a strong authentication for a selected amount of users (P6, R12). Furthermore, the main instance must not allow direct connections without using logged containers (P8, R13). This prevents privileged users from being able to perform non-logged actions and to access every NF once the attacker has access to the main instance.

Also, a single main instance limits the number of requests the solutions can handle (P7). To solve this, the solution needs to be scalable (R14) and extendable. In case the main instance fails, a fall-back solution is needed. The fall-back solution should not function as a backdoor for attackers (P10) and therefore should be well protected (R15).

The requirements resulting from the current problems with PAM and the available solutions must be covered by a new solution. The Functionality of the new solution is covered in the following sections, together with its fulfillment of the requirements.

5 Solution

The key idea of the solution is to use distinct and purpose-built software containers for every requested connection. A global service, the global orchestrator (GO) instantiates the containers for the requested connection. The user can connect to the container and the container connects to the requested NF.

The relation between users and containers is 1:n, one user can connect to n network functions using n containers, but one container is only accessible by one user (see Figure 1).

Multiple users can access the same network function simultaneously, but with different containers, distinct from each other to separate the connections from each other.

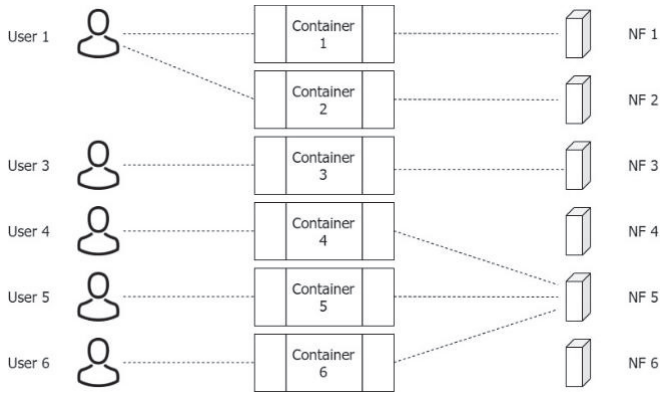

Figure 1 Overall idea to connect users to network functions (NF) using containers.

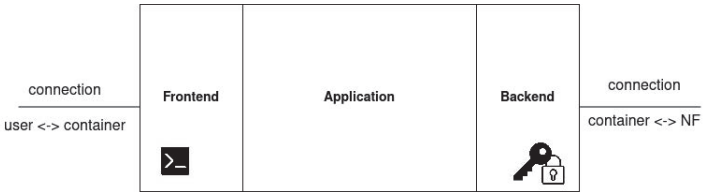

Figure 2 Architecture of a container.

5.1 Software Container

The software containers are instantiated and running in a container cluster. In order to establish connections and forward commands, they have a simple architecture consisting of three layers: frontend, application layer, and backend (see Figure 2). These layers are explained in the following.

Frontend

The frontend is the interface for the user to connect to the container (see Figure 2). The frontend is build depending on the purpose and NF. It could contain a virtual terminal, a login, and a logout function. It could also contain a complete web application if required.
The virtual terminal is the interface for the user to type in the commands he wants to execute on the NF. It might also have an integration for needed authentication methods and single-sign-on integrations.
The connection between user and container can be HTTPS when a virtual terminal is used. Otherwise, SSH, telnet, VNC or RDP can be used. In that case, a virtual terminal is not needed, the container just forwards the commands to the NF.

Application layer

The application layer is the middle layer between frontend and backend (see Figure 2) and can be configured depending on the requested connection.
When a virtual terminal is used in the frontend, the commands have to be formatted into the desired connection type between container and network element. This can be done with a command interpreter in the application layer, translating the command from the terminal into the format the NF supports.
Also when other connections are used between user and container than between container and NF, the commands might have to be interpreted.
Another important function the application layer provides is logging. All actions executed in the container and every command have to be logged. Pushing the logs to a logging backend will be done by a separate logging container.
The application layer sends the required information directly to the logging container where it gets pushed to a logging backend.
Other functions can be added to the application layer if needed, such as a filter to block certain commands that are not allowed as shutting down network functions.

Backend

The backend establishes the connection from the container itself to the requested NF. The kind of connection depends on what the NF is configured for.
In order to connect to the NF, the container backend needs credentials to authenticate against the NF. Different types of credentials can be used, such as signed certificates, digital keys or passwords. The credentials will be saved in the backend when the container is instantiated and always have an expiry date. Furthermore, the backend also has a daemon for the chosen connection type installed, such as a SSH daemon for SSH connections.

5.2 Container Cluster

Containers used for the connections have to run in a so-called container cluster.

The general container orchestrator (GO) is responsible for instantiating and maintaining the desired state of the container within the cluster.

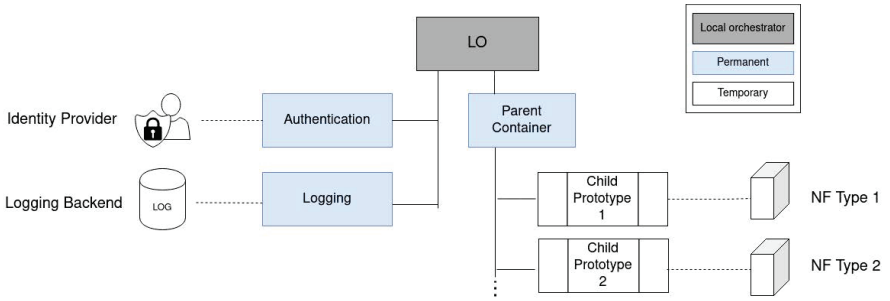

Figure 3 Example of a container with permanent and temporary containers.

Its main task is to instantiate customized containers for every connection request from a user. To achieve that, the GO needs pre- configured container prototypes. These abstract containers can be customized for the requested connection and NF type.

Customized containers can store data such as credentials, username and role information or integrations for different standards as connections protocols, authentication methods or logging databases.

Next to the container prototypes, the CO needs additional containers. While the containers are supposed to live temporarily, other functions might be needed permanently (see Figure 3).

Examples for permanently needed functions are authentication functions and logging functions.

The authentication and logging shall not be defined in the containers themselves, as containers should always be instantiated with only one application to make the architecture modular. Also, the separation allows to maintain different kinds of authentication methods and supported logging frameworks as they would be configured in every single container.

The master container of the GO needs an interface to receive requests for instantiating a customized container.

5.3 Connection Establishment

Figure 4 shows the process of a connection establishment.

In the context of a network, a global orchestrator (GO) is needed to organize all needed actions in a centralized fashion. A certificate authority (CA) signs generated credentials. A local orchestrator (LO) is needed to instantiate, maintain and delete the containers inside a container cluster. The

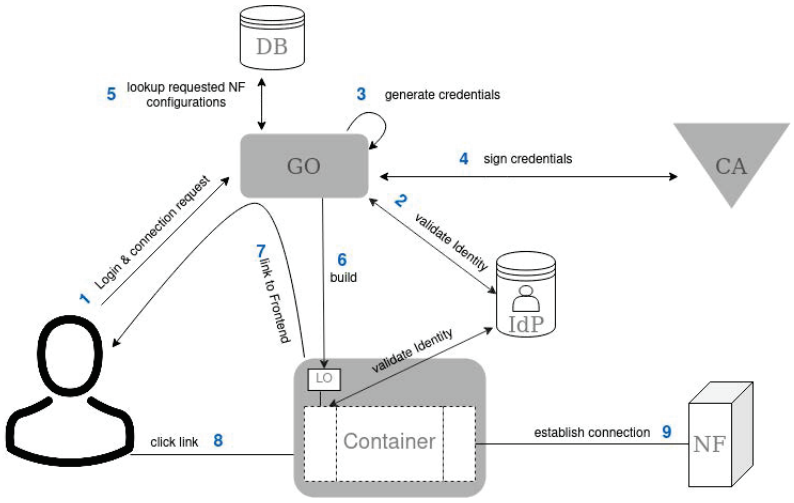

Figure 4 Process of connection establishment using a container with HTTPS frontend.

network element trusts the CA and accepts credentials that are signed by the CA (see Figure 4). The message sequence is explained below.

(1) When a user wants to connect to an NF, he has to login to the GO and creates a connection request for a specific element. (2) The GO then validates the identity of the user. In order to instantiate a container customized for the user, (3) the GO looks up the specifications of the NF from a database (DB) for the network element including the network type, accepted credentials and whatever is needed to instantiate the customized container. (4) Credentials are generated and (5) get signed by a certificate authority. (6) The GO then triggers the LO to instantiate a container for connecting the user with the requested network and (7) sends back a link to the GO. The GO forwards the link to the user. (8) The user can click on the link and gets access to the container. (9) Then the container establishes the connection to the NF and uses the credentials signed by the CA to authenticate against the NF. Trusting the CA, the NE accepts the connection request and establishes the connection.

6 Discussion and Solution Analysis

In this section the solution is analyzed with respect to the requirements given in Section 4. Solution specifications are referred to the Requirements using the abbreviations from Section 4. Discussion on other benefits from the solution are also presented in this section.

Since containers are used for every requested connection, the connections become distinct from each other (R1).

Customizing the container to the users privileges in the building process covers (R2). With the logging function of the application layer the solution allows a complete logging (R3). But the solution itself doesn't provide a tamper-proof logging backend (R4), only the interface to push logs to a logging backend. Therefore, the requirement (R4) is partly fulfilled by the solution and needs to be supplemented by a centralized logging backend and forensic tools to analyze log files.

MFA can be implemented between user and frontend (R5). Credential rotation is not necessary as the credentials in the container backend are generated for every new container (R6). The containers connecting administrators to the GO can require a stronger authentication method (R12) to avoid attackers from compromising the GO. In addition, a user connected to the main instance can't connect to the NFs directly without using containers (R13). Therefore, all actions performed on the main instance itself get logged.

Similarly, the GO can be configured to not allow direct connections to AFs (R7). However, it is the responsibility of the cloud provider to remove local credentials from the NF and remove workarounds such as direct SSH (R8, R9). This prevents attackers from bypassing the logged containers.

The modular architecture allows developers to add container prototypes to adapt them to new NFs or to enable different authentication methods without affecting others (R11). In this way the container prototype for accessing the GO can define a stronger authentication method (P12) This solution can be reused across different software layers (R10).

Scalability for the containers is given by the nature of containers themselves (R15). Of course, the hardware has to provide enough resources to generate the amount of containers required to allow all requested connections. But next to the containers, also the GO has to be scalable. To solve this, multiple GOs can exist in a network or even in other networks. This allows full scalability and also provides a fallback solution in case one or more GOs fail. This in combination with (R12) provides a well-protected fall-back solution (R15).

But not only the access should be secure, but also the solutions itself. One factor is the needed security of the container. For example, an attacker should not be able to change the NF or user role after the container is instantiated. Also, the GO should not be manipulable. As the GO initiates the building procedure of the container, it could be misused by attackers to instantiate or manipulate the building process of a container. Therefore, access to the GO

has to be strictly restricted and protected. The API used by the user should be clearly defined and well-tested.

Another problem comes with the availability of the GO. Implementing only one GO for handling requests, the availability is again depending on the availability of a single point and scalability limited. Thus, the GO should be multiplicable to handle huge amounts of requests, big data or a denial of service attack.

When the solution is used for a network or service dealing with huge data or a vast number of transactions giving access, transporting data and logging actions becomes an issue.

To deal with this issue efficiently, the containers for this solution need to be as small as possible. To achieve that, the base image for the container has to be small, every feature that is not necessary for the main functions or security purposes has to be deleted. Also the software in the application layer has to be as slim as possible. This leads to the fact, that a container with HTTPS frontend with a virtual terminal and a command interpreter in the application layer results in a size of container that might be too big for big data purposes. Therefore, containers for this purpose should only forward commands and avoid any unnecessary overhead.

In conclusion, the PAM solution needs to provide a scalable and customizable implementation, flexible enough for an always changing environment and ready for big data purposes.

7 Future Work

Future research should define more details of the solution architecture. A prototype should be developed and tested in order to find the issues of this solution focusing on the existing problems and requirements analyzed in this paper.

8 Conclusion

For PAM in cloud environments are still no secure solutions available. Problems with the currently used methods are credential misuse and problems with the number of different user groups and the different configurations of the solutions in the cloud.

The currently available solutions all use a similar approach and handle all connections with a single proxy or jump server. Problems are mainly scalability, weak fall-back solutions and existing workarounds.

The approach of using containers for each requested connection solves these problems. The connections of the individual users are separated from each other and every action is logged without exceptions. Furthermore, the concept can be implemented in a scalable way and also takes fall back solutions into account.

If the functions are limited and the solution is implemented in a lean way, it is also suitable for big data applications. Therefore, the idea of using containers for PAM is a viable approach, which should be implemented and tested after a more detailed conceptual design.

References

[1] R. Kneuper, "Software Processes and Life Cycle Models. An Introduction to Modelling, Using and Managing Agile, Plan-Driven and Hybrid Processes", Springer, 2018, ISBN 978-3-319-98844-3.

[2] M. Steinhoff, "About problems and requirements with privileged access and authorization management in cloud-based multi-tenant networks", 2020, In: Proceeding: International Symposium on 5G & Beyond for Rural Upliftment 2020. e-ISBN: 9788770222174 doi: https://doi.org/10.13052/rp-9788770222174.

[3] Verizon, "2019 Data Breach Investigations Report", 2019.

[4] H. Schulze, "2019 Cloud Security Report", ISC2, 2019.

[5] Gartner, "Magic Quadrant for Privileged Access Management", 2018. ID G00356017

[6] CyberArk, "Privileged Access Security", version 11.2, 2020. Url = https://docs.cyberark.com/Product-Doc/OnlineHelp/PAS/Latest/en/Content/PASIMP/Introducing-the-Privileged-Account-Security-Solution-Intro.html, 2020-02-21.

[7] BeyondTrust, "Prvileged Remote Access", 2020. Url = https://www.beyondtrust.com/de/remote-access, 2020-02-21.

[8] Centrify, "Centrify Privileged Access Service", 2020. Url = https://www.centrify.com/privileged-access-management/privileged-access-service/, 2020-02-21.

[9] SSH Communications Security Inc., "PrivX® – lean, modern privileged access management", 2020. Url = https://www.ssh.com/products/privx, 2020-02-27.

[10] SSH Communications Security Inc., “Web Access Architecture”, 2020. Url = https://help.ssh.com/support/solutions/articles/36000166941-web-access-architecture, 2020-02-27.
[11] Ping Identity, “PingAccess. Access security for apps and APIs”, 2020, Url = https://www.pingidentity.com/en/software/pingaccess.html 2020-02-27
[12] OWASP, “Owasp Top 10”, report, OWASP foundation, 2017.
[13] Cloud Security Alliance, “Top Threats to Cloud Computing: Egregious Eleven”, Report, 2019.
[14] D. Catteddu, G. Hogben, “Cloud computing - benefits, risks and recommendations for information security, report, ENISA, 2012.
[15] M. Iorga, “Challenging security requirements for us government cloud computing adoption, 2012.

Biography

Marleen Steinhoff is a bachelor student at the Munich University of Applied Sciences since autumn 2017 and will receive her B.Sc. in Information Systems and Management in March 2021. She worked as an intern at Rakuten Mobile Inc. in Tokyo, Japan on access management for cloud-based 5G networks. Since September 2020, she acquires experience in incidence response automation while working at Siemens. Her bachelor thesis at Siemens covers the handling of cyber security playbooks for the automation of incidence response in networks.