Patterns & Anti-Patterns for Zero Trust Access Management

Teleport

CONTENTS

INTRODUCTION

This guide addresses how IT organizations can optimally manage access to modern cloud server fleets. The elastic nature of cloud infrastructure often requires IT to deal with thousands or even millions of nodes, with VMs that are launched and deleted frequently. In addition, the requirements to access these infrastructures also change frequently, and can involve people both inside and outside the organization. Under these conditions, implementing a system of Zero Trust Access Management that meets today's requirements for scalability while ensuring regulatory compliance and enforcing security can be a daunting challenge.

This guide will help. It provide guidance on patterns and anti-patterns that we have seen implemented by system administrators building access management systems on top of OpenSSH systems.

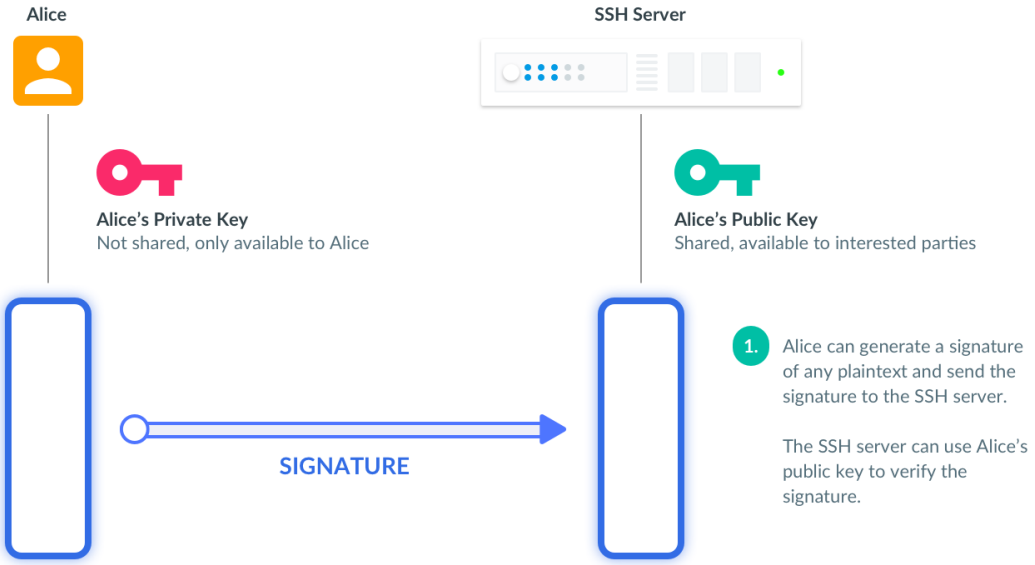
We adopted many of the SSH infrastructure patterns mentioned here while building our Teleport product, a modern SSH server that manages privileged access to elastic infrastructures without creating more problems than it solves. But before we dive into the details of the access management and SSH patterns and anti-patterns, let's cover the typical key components that are used as building blocks of an infrastructure access management system using SSH.

BACKGROUND OF ACCESS MANAGEMENT COMPONENTS

SSH

SSH stands for Secure Shell, a cryptographic network protocol widely adopted in the internet to access servers via remote logins. Access is achieved via an SSH client that dials into an SSH server.

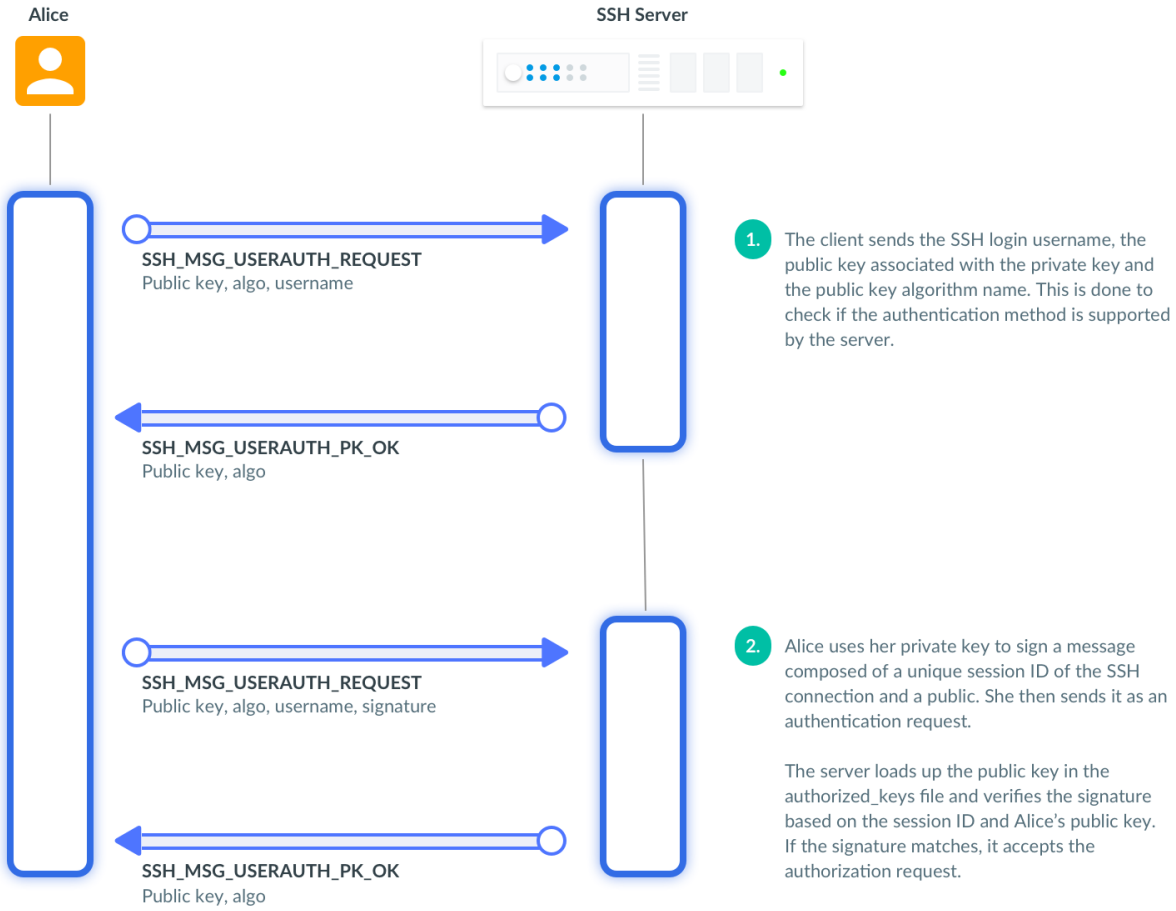
SSH Public Key Authentication



SSH keys are based on Public Key Cryptography. The public key is distributed openly, and anyone holding it can encrypt data. However, only the owner of a private key can decrypt that data. With this method, a user owns and never shares the SSH private key and SSH servers are set up to trust the identity of those users who can provide a trusted private key.

Private keys can also be used to sign any message where verification requires only the public key. Public key signatures are used in SSH public key authentication. It's not necessary to send the plaintext that was signed alongside the signature assuming both parties already know the plaintext that was pre-negotiated.

Here are more detailed steps of public key authentication:



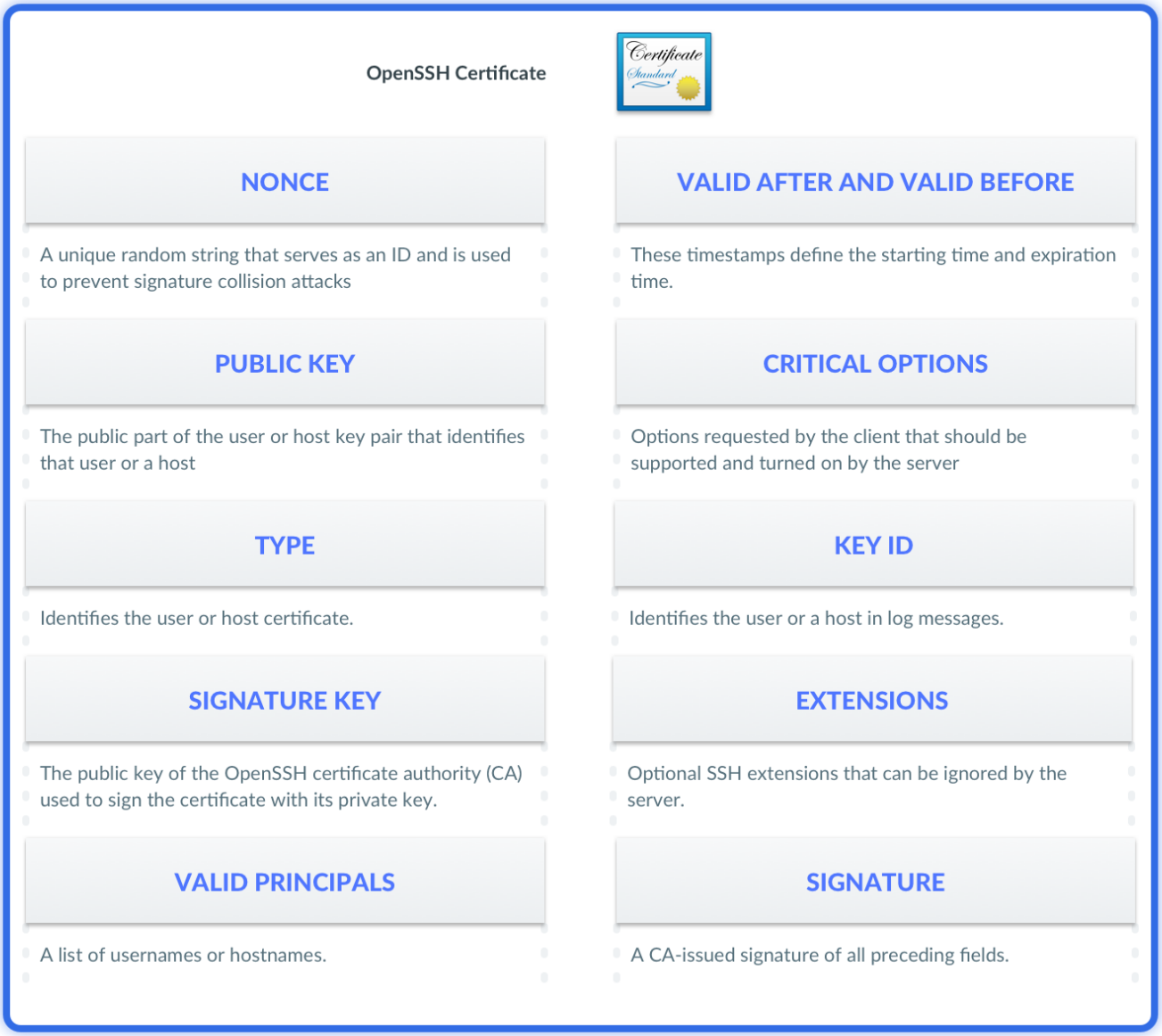
You can read more here.

OpenSSH Certificates

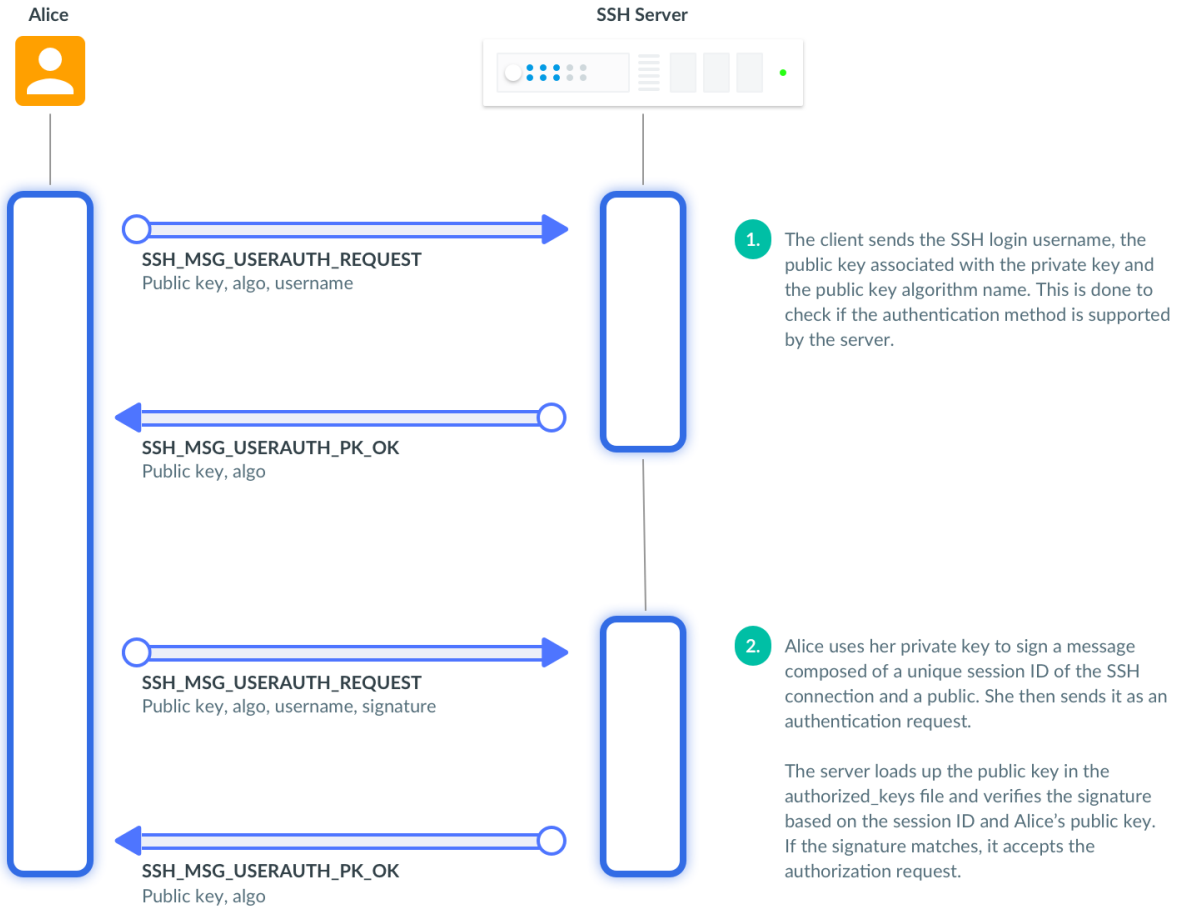
The OpenSSH certificate is a special extension built by the OpenSSH team. OpenSSH certificates are built using public keys. The OpenSSH certificate authority (CA) is a special trusted party that holds its own public and private key pair.

OpenSSH CA keys are not used to authenticate, but to sign SSH certificates. An OpenSSH certificate consists of a collection of fields (listed in the chart) signed by the CA.

OpenSSH Certificate Authentication



OpenSSH certificate authentication extends public-key-based authentication and uses the same protocol messages:

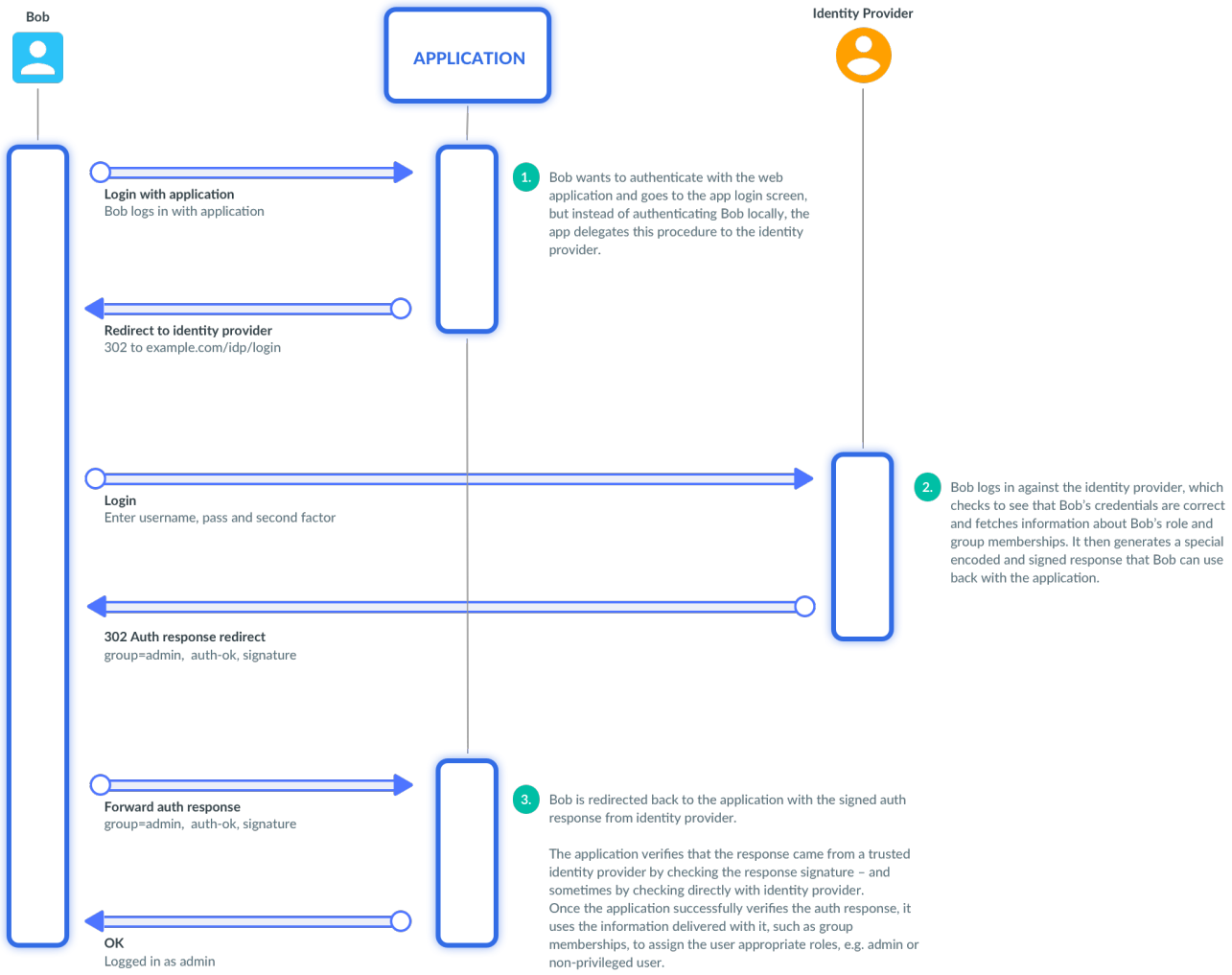


LDAP

LDAP is a directory access protocol widely used in enterprise server deployments. LDAP servers provide information about user and group memberships and sometimes are used to store users' passwords and SSH public keys.

Identity Provider

Identity provider (IdP) is a system that manages centralized identity for all principals (users or computers). Modern identity providers are used to set up a single sign-on (SSO) service for many computer systems.



Identity providers are widely used in the enterprise and getting more traction in smaller teams because they allow organizations to provide internal authentication service, allowing different applications to offload the process of authentication to the centralized service. Providers include Okta, Auth0, ADFS and many others.

There are special protocols that allow applications to integrate between identity providers. In our experience, the most widely used one is SAML 2.0. However, OIDC is getting more traction, as it provides easier authentication flows for mobile applications.

AUTHENTICATION AND AUTHORIZATION

These next three sections go through some best practices for Zero Trust Access Management. “Production Patterns” are methods we have seen that are useful in production and sometimes necessary to pass compliance audits. “Anti-patterns” are generally methods you want to avoid in production as they can lead to problems at scale.

While some of the methods listed here as anti-patterns could be good solutions for small scale server deployments, they will quickly fall apart with larger clusters and organizations. Sometimes an anti-pattern works well in theory, but its practical implementation is usually problematic.

Anti-Pattern: Host-Based Authentication

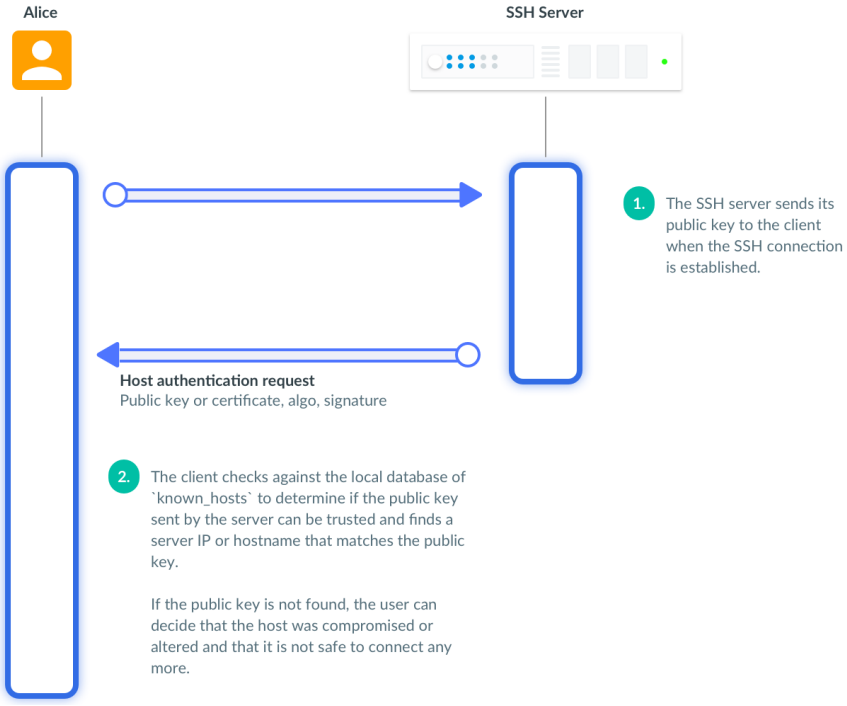
With the host-based authentication method, the client sends a signature and public key that are signed by the host, not the user. The Achilles heel of this approach is the behavior of a compromised host.

- A compromised host can impersonate other hosts in the system, so servers have to add extra checks over an insecure network.
- Users on compromised hosts can impersonate almost any other user allowed to login from that host.

This authentication method should be disabled at all times for SSH servers.

Anti-Pattern: Trust on First Use

When the SSH connection is first established, the SSH server sends its public key to identify itself to the user as shown below.



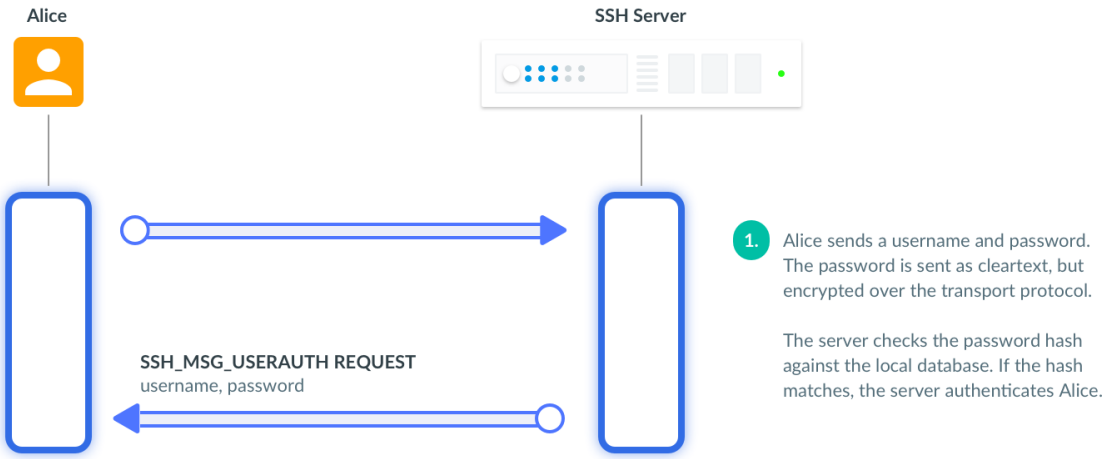
One suggested approach to this situation is "trust on first use," i.e. assume that the host is trusted if it is the first time that host has been encountered. This method assumes that the IP or hostname has not changed, and therefore the combination of hostname and public key can be trusted, resulting in this warning:

```
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@    WARNING: REMOTE HOST IDENTIFICATION HAS CHANGED!    @
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
IT IS POSSIBLE THAT SOMEONE IS DOING SOMETHING NASTY!
Someone could be eavesdropping on you right now (man-in-the-middle attack)!
It is also possible that a host key has just been changed.
The fingerprint for the RSA key sent by the remote host is
51:22:00:1c:1e:6f:ac:ac:de:f1:53:02:1c:7d:35:68.
Please contact your system administrator.
Add correct host key in /home/example.com/.ssh/known_hosts to get rid of this message.
Offending RSA key in /home/example.com/.ssh/known_hosts:15
RSA host key for 192.168.1.1 has changed and you have requested strict checking.
Host key verification failed.
```

At this point, the user can either alert the security team and raise an incident or ignore the warning by removing the old key. For cloud environments, however, the same IP address and internal host name can be reused many times. That's why we recommend against using only host public keys for checking real host identity.

Anti-Pattern: Using Password-Based Authentication

The diagram below shows how password-based authentication works.



While theoretically there is nothing wrong with this approach for small scale servers, we recommend against it because it will likely break as an organization's infrastructure footprint grows. Here are just some of the problems we have encountered:

- Password authentication invites brute-force attacks by bots that pollute the logs, so admins are forced to set up tools like fail2ban to prevent excessive scans.
- The user experience degrades because users have to re-enter the same password many times on every server they log into, forcing them to copy and paste passwords or use very simple and short passwords.
- In the case of per-environment passwords, users tend to re-use the same password. If that's prohibited and checked, they come up with workarounds like using sticky notes, a shared password for the team, or overly simplified passwords.
- When local passwords expire, some systems are configured to prompt users to change their password to continue. This confuses many users, as they are not ready to come up with a good replacement password and tend to forget it if they do. Many times, users are not sure if they have changed their password for a single server or cluster-wide, as the system does not inform them about this.

Anti-Pattern: Shared Private Key

When provisioning an instance, cloud providers like AWS ask for a pre-installed public key to set up as a key for first access to the server.

Teams are tempted to use the same private shared key for every single instance. This is a bad practice. Too many people will have access to the same key, so the security of the whole company will rely on a single secret.

A simple and better alternative is to set up instances to boot with pre-installed trust to an external authentication method like an SSH user CA. We suggest using a shared key only as a last resort. It should be encrypted and stored in a safe place.

Anti-Pattern: Using LDAP with Password Authentication

In addition to suffering from the general problems of password-based authentication, there is another problem we have encountered with LDAP password-based authentication: LDAP setup becomes a single point of failure for the entire cluster. If the LDAP server or servers become overloaded or go down, the whole cluster goes down.

In theory it is possible to set up a highly available and scalable LDAP server (e.g. FreeIPA). However, in practice we have seen this fail many times and we recommend against it for large server deployments.

Anti-Pattern: Unknown Source of User Identity

When using public SSH key authentication, we have encountered many situations when it was not possible to determine why a certain public SSH key was in authorized_keys, or even to determine the origin of the key in the first place.

This will be a problem during any security audit or forensics analysis of the server. As a rule of thumb, we recommend a single source of truth for all public keys or certificates, periodically synced on the hosts, with manual changes to the servers prohibited. The single source of truth can be as simple as an S3 bucket with public keys or a Github repository that is periodically synced up by an Ansible job.

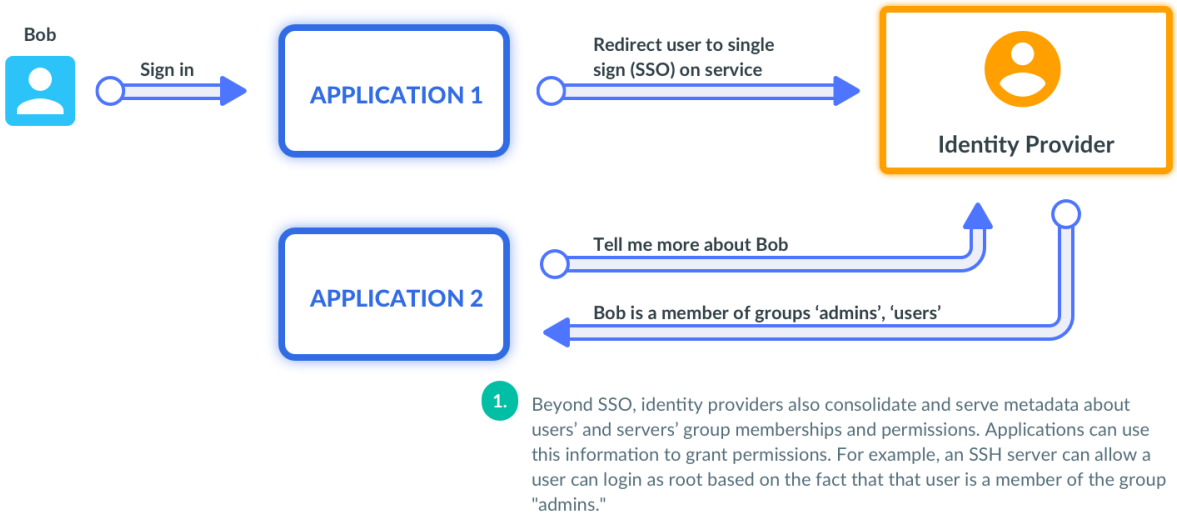
Production Pattern: Using LDAP to Set Up Trusted User and CA Keys

LDAP can be used to store users' public keys or SSH CA keys. In this method, every host should have an agent that connects to the LDAP server and syncs public keys on a regular basis. This method has several advantages over password-based authentication.

- It does not require the LDAP server to be up and running at all times, because the updates are periodic. Therefore, authentication works even if the server is down for some length of time.
- Only the non-secret public keys are stored on the LDAP server, reducing the encryption requirements for secrets at rest and simplifying deployments.
- This method provides a single source of truth, as users are managed through the identity provider. Keycloak and FreeIPA work well together to implement this pattern for a distributed infrastructure.

Production Pattern: Identity Providers

Identity providers solve many problems for medium- and large-sized organizations. They provide a Signle Sign-on ("SSO") service, so applications don't have to use local and custom authentication methods but instead can rely on a centralized service.



1. Beyond SSO, identity providers also consolidate and serve metadata about users' and servers' group memberships and permissions. Applications can use this information to grant permissions. For example, an SSH server can allow a user can login as root based on the fact that that user is a member of the group "admins."

Production Pattern: Second Factor Authentications with Identity Providers

While identity providers are recommended, they have a significant drawback: The whole organization is only as secure as the password of a privileged user of the identity provider. This means that a single password solution is no longer a secure option. A strong second factor authentication is necessary, one that is hard to get without physical access such as FIDO U2F keys. Using SMS as a second factor is not a good choice because it is relatively easy to hack.

Anti-Pattern: Local or Poorly Managed Second Factors

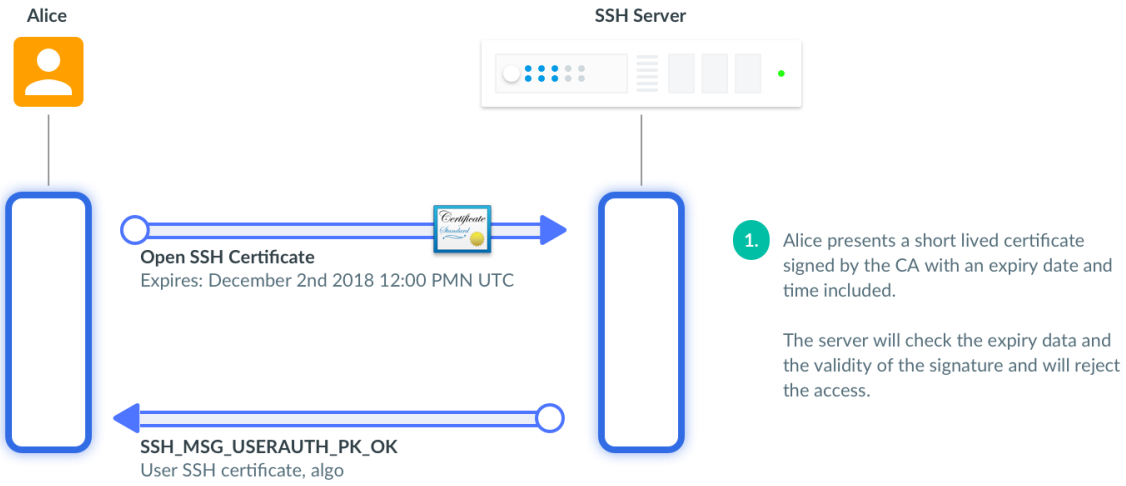
Some administrators deploy a local second factor solution where the second factor is enforced locally on a bastion server. This can cause problems because the private material of the secret factor is usually neither encrypted, nor is it stored securely, backed up and replicated.

This method is confusing for users who don't have a good way to manage their second factor. Also, if server data gets lost, everyone has to re-register with the system. This can make the second factor a nuisance and leads users to start developing workarounds.

Production Pattern: Short Lived User OpenSSH Certificates

OpenSSH certificates are very scalable because servers can check whether they trust the user based on just one signature from the trusted CA. This means that servers do not need to be connected to the active authentication server at all times.

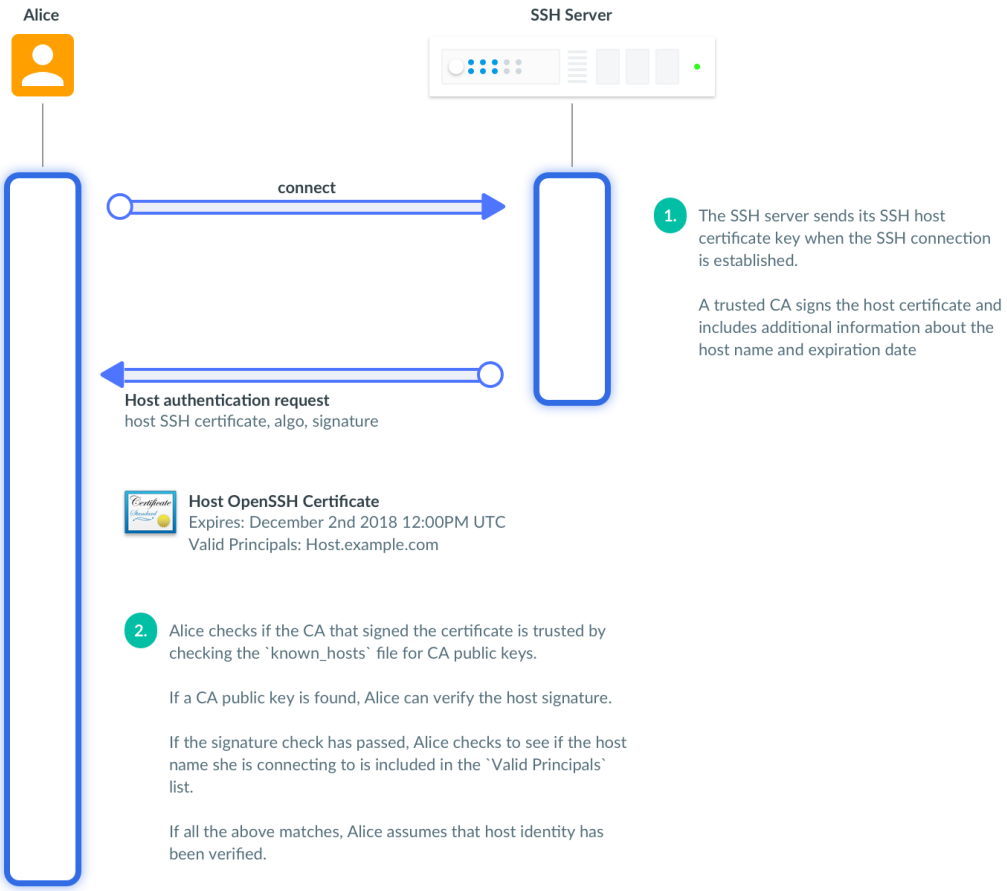
This method does create one problem: revoking the access for an already issued certificate when an employee leaves the company. Fortunately, OpenSSH Certificates include an optional expiry date that can be verified by the server in addition to the signature.



Organizations can issue certificates that are only good for a few hours before they auto-expire (without any additional action). From a security perspective, the shorter the duration for these certificates the better. Ideally, they should be issued only for the duration of the session. However, in practice, several hours or the duration of the work day may be used for convenience.

Production Pattern: Host OpenSSH Certificates

As with a public key authentication, when an SSH connection is established, the host can send a signed SSH certificate to the client to verify the host's identity. The host certificate is signed by the trusted CA. It includes information about the host name, and has an expiration date. Here is the procedure if Alice needs to check if she can trust the certificate:



This method has several notable differences from the public key method.

- The client does not need to store public keys for every host it connects to, only for a list of trusted CAs.
- Additional information is available concerning valid principals that allows the client to check if the hostname or IP matches the certificate. This makes it harder for the host to impersonate another host.
- If the signature check has failed or the CA is not trusted, this usually means that some serious misconfiguration has happened or that someone is attempting a man-in-the-middle attack, in which case the security team should be alerted.
- Even if the public key of the host has been changed because the hostname has been reused in the cloud environment during instance re-provisioning, the certificate will still match, as there will be no conflict between different public keys.

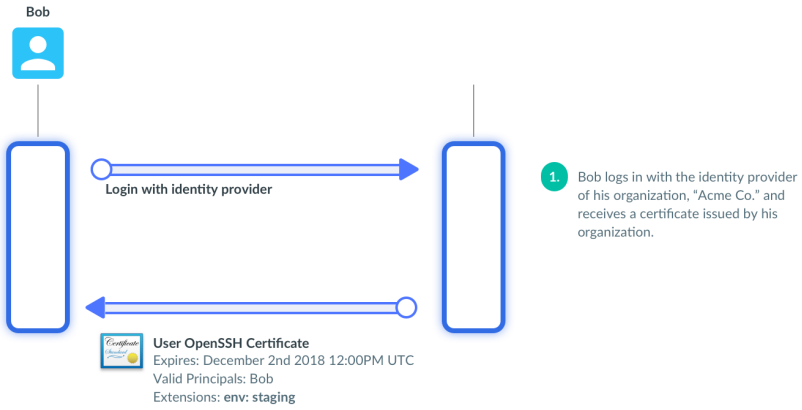
Anti-Pattern: Forever Certificates

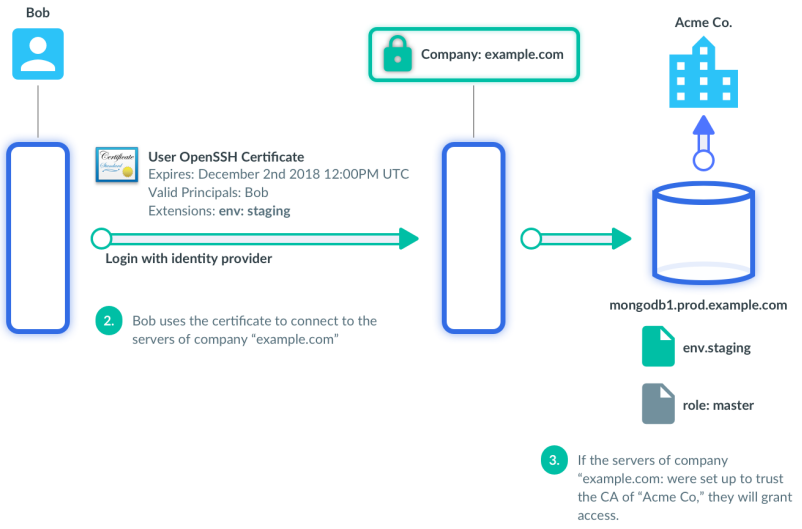
We recommend against issuing certificates without expiry date because it is very difficult to make a certificate invalid if the host has been compromised or an employee's laptop has been stolen.

Web certificates provide a feature called Certificate Revocation Lists (CRL). However, this feature has been criticized by the security community for its flaws. With OpenSSH certificates, there is no CRL infrastructure, so the only way to invalidate a certificate issued by the system is to invalidate the CA, which could be very disruptive.

Production Pattern: Inter-Org Trust

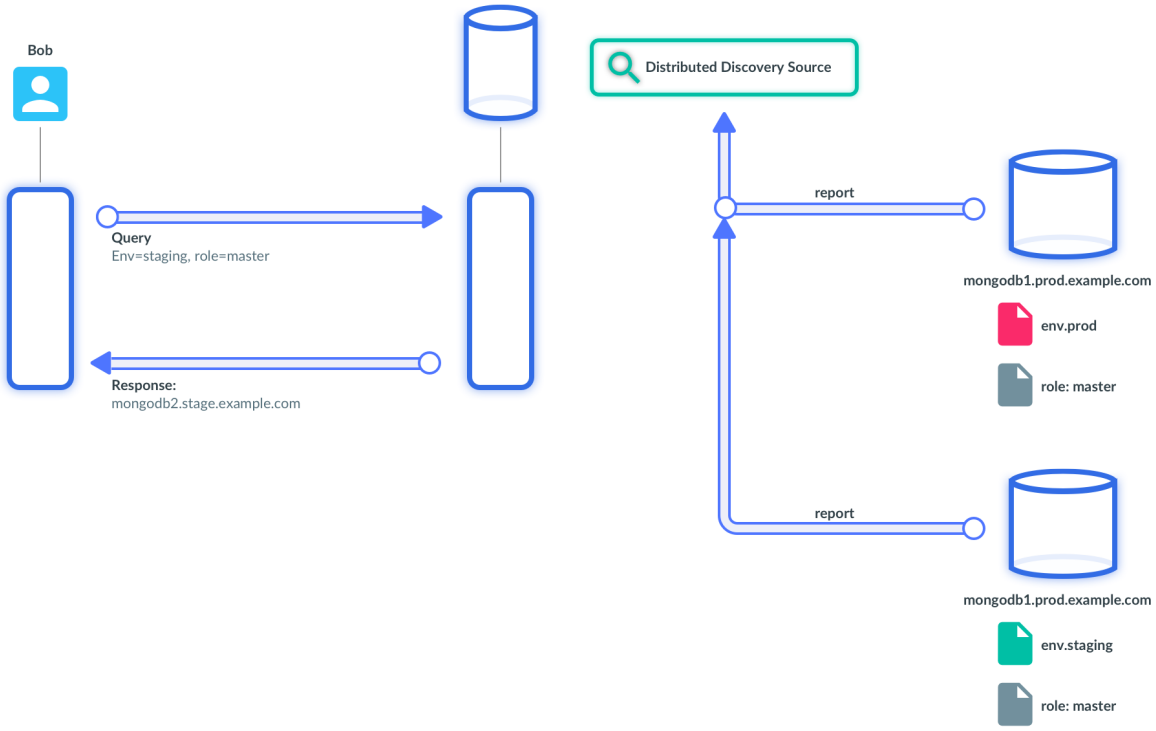
There are cases when you need to set up access for an external organization to your infrastructure (e.g. for a security audit). The CA approach makes this fairly easy, as long as the access management layer supports "external" CAs. Users in the external organization can use their credentials to access servers on the external infrastructure, as long as servers are set up to "trust" the key of the other organization's CA. (In this case, the external auditor's organization will issue certificates for their users as usual.)





Production Pattern: Centralized Discovery

Modern infrastructure fleets are elastic. VMs and cloud instances are added and removed on an hourly basis. Also, HA databases are performing automatic failover and changing cluster membership, so it becomes critical to maintain an accurate, up-to date view of the infrastructure.



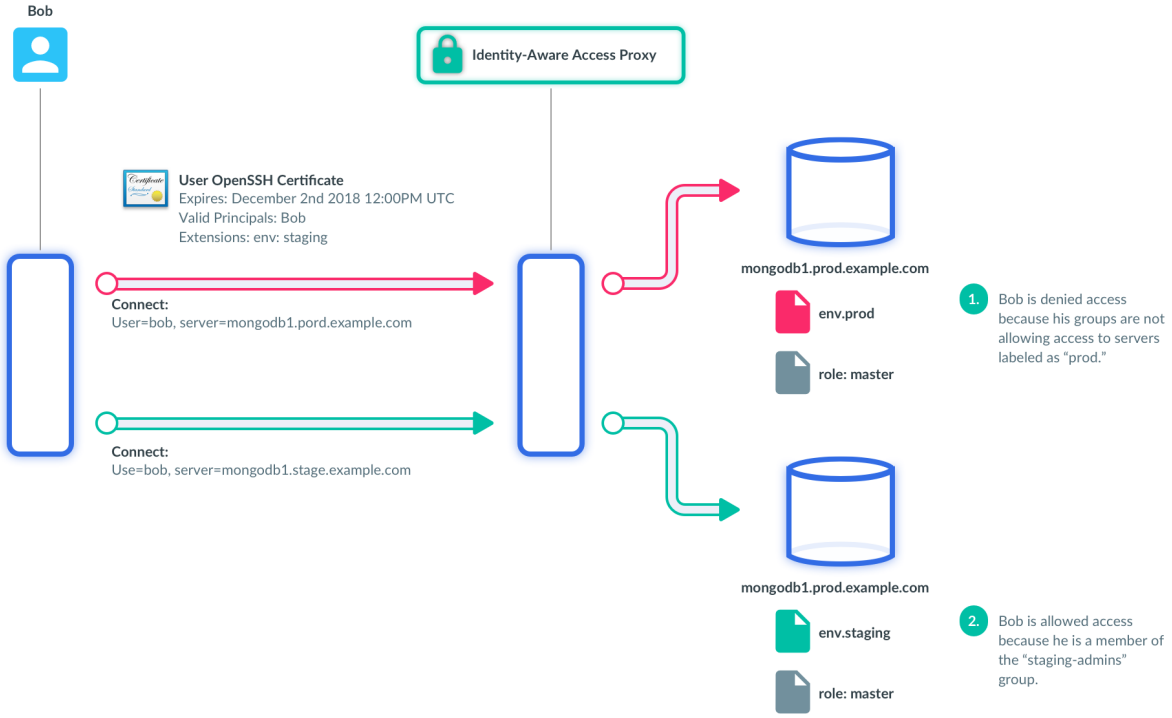
Modern access management tools can integrate with discovery services to help administrators and users segment access based on reliable information about the services and servers.

Scattered, wiki-powered knowledge about servers becomes obsolete and can pose security risks as well. Imagine a user trying to login using an IP address that had been recycled to another tenant in the cloud. Combined with the trust of first use anti-pattern, this could result in a successful phishing attack by a bad actor.

Production Pattern: Role-Based Access Control

The use of an identity provider, combined with centralized discovery, allows role-based access control (RBAC).

The identity provider is responsible for issuing an OpenSSH certificate that includes user identity and group membership information. An identity-aware access proxy can use this information to allow or deny access to different parts of the infrastructure.



Instead of setting up access for every server for every user individually, users can be granted access to parts of the infrastructure, as in the example above.

Dynamic discovery enables an even more powerful pattern: granting access based on the dynamic traits of users and the servers they need to access. For example, some users can be given access only to database replicas.

MONITORING AND ALERTING

Production Pattern: Centralized Audit Logging

If a potential security breach is discovered on one of the servers in a cloud infrastructure, determining the scope of the attack is crucial. If it was localized and did not involve any production data, recycling the servers at risk would be an adequate remediation. If global, an expensive full infrastructure audit would be required.

In the absence of a centralized security audit log, making this determination requires inspecting every single server in the company to check for traits resembling the attacker, e.g. a suspicious source IP or suspicious access patterns. This could take several weeks and require every single department to manually collect and submit logs on a crash basis.

With a centralized log, the process is much simpler and less painful. Centralization does not necessarily require buying expensive commercial logging and security scanning software. A good first step could be collecting auditd logs to a centralized location with syslog.

Production Pattern: Anomaly Detection and Alerting

Having centralized and structured data regarding events such as failed and successful logins and operations performed on a server helps security teams more easily detect anomalies on the infrastructure. For example, if audit data is combined with CPU and network usage data, it makes it easier to watch for attempts to extract data from the servers.

Anti-Pattern: Intercepting Interactive Commands

Special command detectors lock users out of the system when they encounter specific commands typed by the user such as “reboot" or "rm." These scanners are implemented at the protocol level and do simple pattern matching, so they are very easy to work around...

```
$(echo ZWNobyBoYWNrZWQK | base64 -d)
hacked
```

...and only make life of the administrator harder. Instead, system administrators should rely on SELinux policies, linux user and group membership security.

Production Pattern: Session Capture

It is possible and beneficial to capture the SSH sessions of every user in the system. We don't think that this is a security feature because it is impossible to intercept and reliably understand the behavior of a live interactive session. (It may, however, be useful for forensic analysis of a breached system.)

The benefit of session capture is that it increases transparency in the team, allowing everyone to understand what other team member are doing at any given moment. It also helps during root cause analysis sessions, and can be used to help train new employees.

Production Pattern: Rate Limiting and User Locking

When a proper authentication method is used, brute force attacks will not produce any results. It is simply not possible with current technology to guess a user's private key. Nonetheless, rate limiting is necessary. Without rate limiting, logs will be polluted by failed authentication attempts and will be harder to inspect. In addition, rate limiting serves as a deterrent to bots.

Rate limiting is not strictly necessary for SSH level security, but it is a good additional measure. It becomes more important for other components of the system that use password-based authentication, for example SAML or OIDC identity providers.

It's important to lock out a SAML or OIDC user after several incorrect login attempts to prevent brute-force attacks, especially if second factor authentication is failing while the password is succeeding. This is often a strong signal that an adversary has access to a user's passwords.

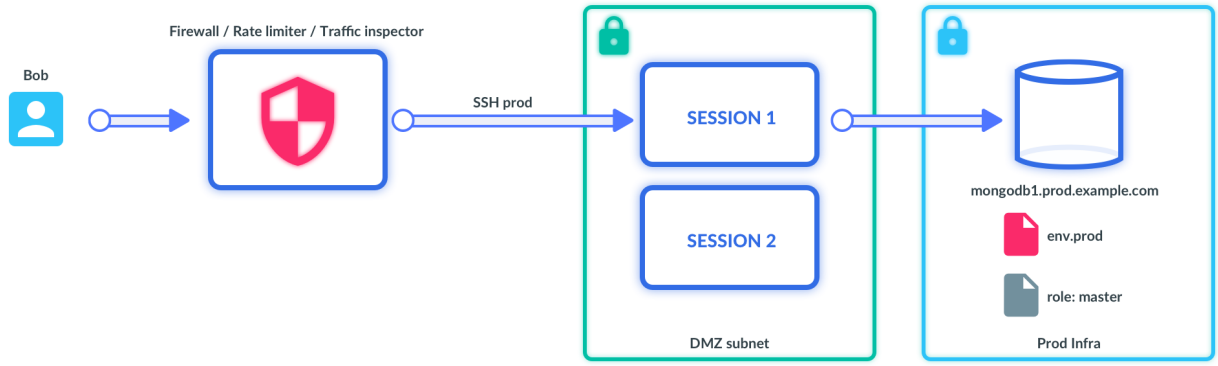
SYSTEM ARCHITECTURE AND MAINTENANCE

Anti-Pattern: Non-Segmented, Publicly Accessible Networks

In theory, if every server is secure and patched, all network communications are encrypted, secure networking protocols are in use and DDOS protection is in place, there is no need to partition a network and build demilitarized zones. In practice, however, maintaining a perfect security setup is extremely difficult. That's why we don't recommend exposing privileged access over the internet to the whole infrastructure.

Production Pattern: Bastion Servers

Bastion servers are access endpoints set up as the only way to get into the rest of the cluster. They provide a convenient way to expose only a small part of the infrastructure. Bastion servers allow a security team to quickly shut down all control plane traffic to the entire infrastructure and terminate all internet connections. If, for example, a laptop capable of extracting data were stolen, its access could easily be blocked.



Anti-Pattern: Source IP-Based Access

Some companies limit the range of source IPs that are allowed to connect to the infrastructure. In practice, we have seen this leads to nothing but confusion, with users adding their home IPs when working from home, clients blocked when a NAT gateway IP has changed, or entire offices locked out because of a firewall misconfiguration. Source IP is not a reliable way to identify the user or the connecting party and we recommend against it for production system access.

Anti-Pattern: Weak SSSH and TLS Set-Ups

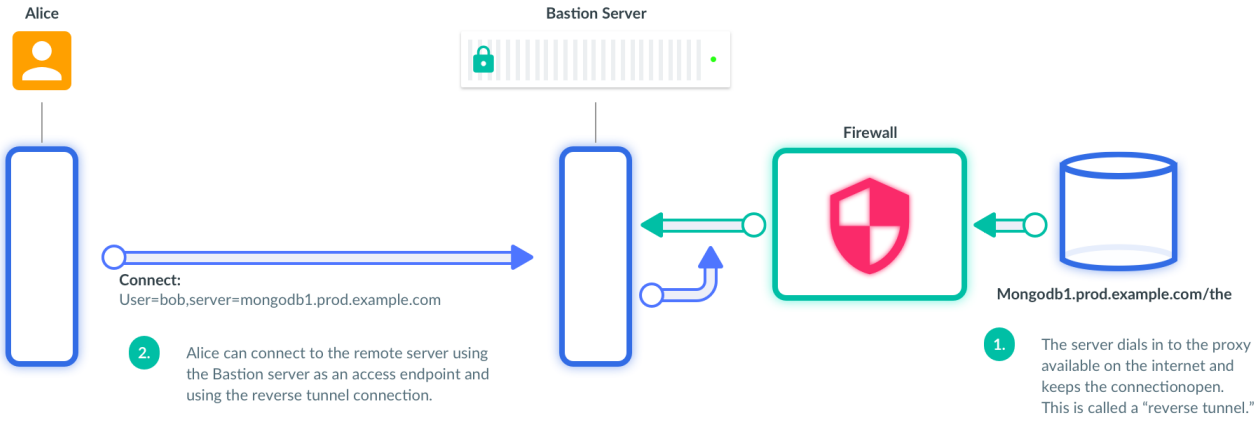
A system is as secure as its weakest link. All security-oriented actions do not matter if SSH is set up to use a cipher with a known vulnerability. We recommend using a set of trusted guides reference supported protocols, cipher suites and key exchange algorithms.

Production Pattern: Idle Timeouts

The system should be set up to lock users' computers after a period of inactivity and also terminate idle connections after a period of time. (A typical idle timeout is about 5 minutes because it does not disrupt the users' daily routine.) This pattern protects users who are prone to mistakes rather than punishing them after the fact.

Production Pattern: Outbound "Reverse Tunnels"

Sometimes is not possible for a server to easily allow inbound internet connections, e.g. in cases where no static publicly available address can be used or when a firewall has been set up to prevent inbound access. In these cases, the "reverse tunnel" method can be used, as shown below.



- Servers can dial into the publicly available proxy and keep the connection – a "reverse tunnel"
- Clients can use this connection as a special tunnel to connect to the target server via a publicly available proxy.

Production Pattern: Certificate Authority Rotation

If an urgent need arises to lock down a system from using OpenSSH certificates to access the cluster, this can be done without seriously disrupting workflow by rotating all certificates in the system to a new certificate authority. In this method, a new certificate authority is created, and all principals in the system receive new credentials.

CONCLUSION

We hope this guide is useful as an overview of Zero Trust Access Management best practices. Teleport, our modern SSH server for teams that manage elastic infrastructure, provides many of these best practices out of the box with minimal operational overhead. Feel free to reach out if you are interested in a demo, or if you'd like to send us comments or feedback.

ABOUT US

This guide was created by Teleport. We build open source software solutions to deliver, access and manage cloud-native applications and infrastructure. Teleport is our security gateway for managing privileged access to server infrastructure through SSH and Kubernetes. Gravity is our Kubernetes appliance to package, publish and deliver cloud-native applications across cloud and on-premises environments. You can reach us at info@goteleport.com. If you're looking to incorporate best practices for zero trust access management, download the open source community edition of Teleport on our site or request a demo of Teleport Enterprise.