Best Practices for Secure Infrastructure Access

Teleport

Best Practices for Secure Infrastructure Access

Abstract

Focus on the resources and employees, not on networks and environments. The usage of firewalls, VPNs, intrusion detection, and other networking components have been the backbone of managing access to important resources. But trends towards decentralization and abstraction has made infrastructure increasingly fluid and enforcement slippery.

The collective shock the world took when all employees abruptly became their own remote office became a tipping point, forcing companies to reconsider how they would remain protected. Doing so provides an opportunity to dig up old roots, prune best practices, and transplant core principles. This whitepaper will highlight modern-day infrastructure challenges, introduce recommended best practices, and apply them to the familiar SSH protocol for greater safety.

Introduction

Technologies build on other technologies to compound growth. It's no coincidence that of the companies with the highest market capitalization within the US, the first non-tech company is the eighth one down, Berkshire Hathaway. Nor is it a coincidence that tech startups can take their valuation into the 10 digits in a flash on the backs of other tech companies. This pace of growth can only be afforded by the innovation of new technologies.

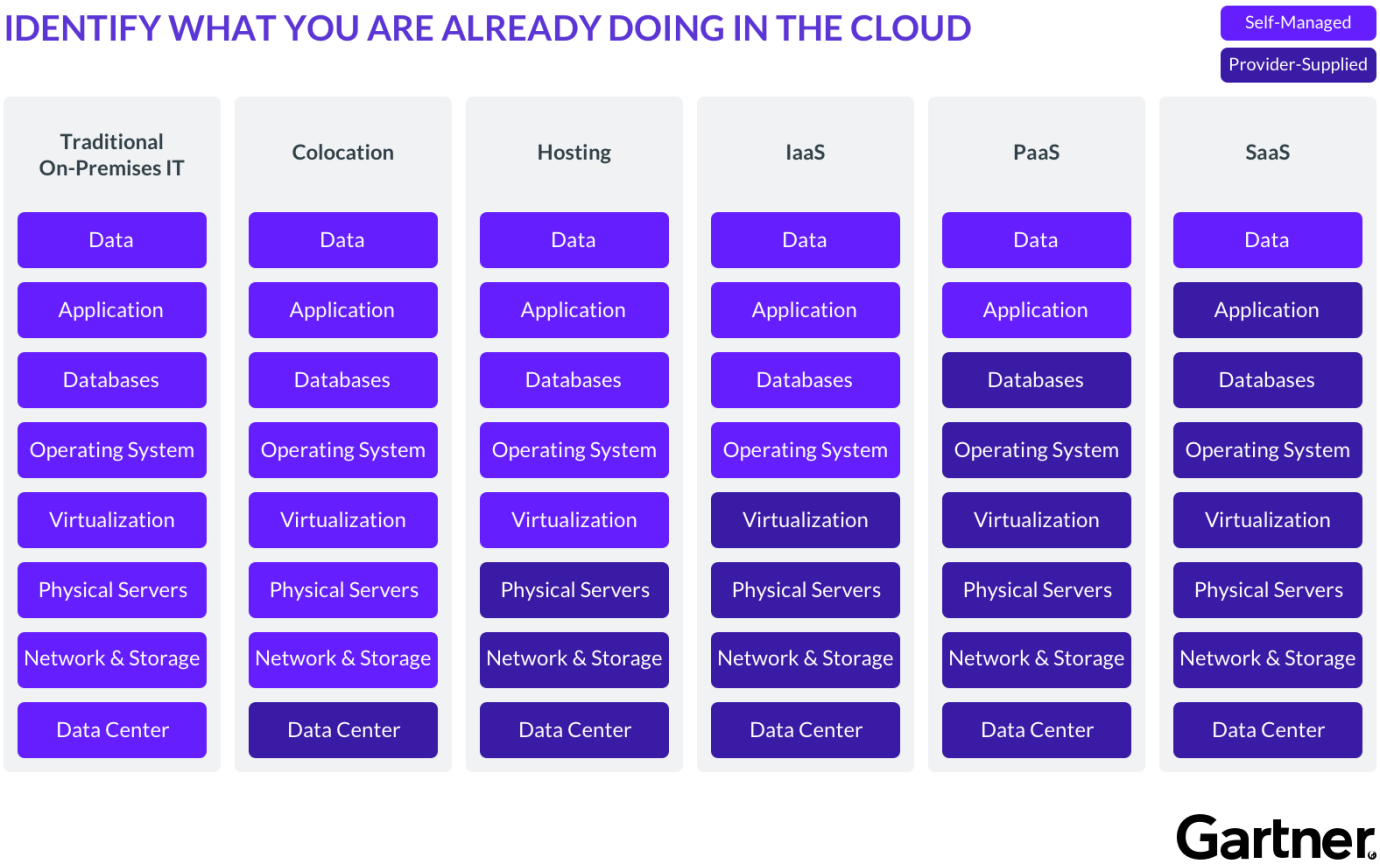
> *"Information technology continues to grow exponentially in size and scale, producing immense complexity."*

Cloud Computing

Over 15 years after the launch of AWS, Gartner still projects 17% industry growth in 2020 to over a quarter trillion in public cloud revenues. The new frontier is no longer centralized server clusters, but at the elusive "Edge," where Gartner predicts 75% of enterprise data will be produced in just five years (2025).

Everything-as-a-Service

The SaaS business model is its own industry, offering every slice of infrastructure as a managed service. Referred to as Everything-as-a-Service (XaaS), these service providers allow companies to outsource core technologies, maintenance, and monitoring at a fraction of the cost required to build in-house.



Source: Gartner

Internet of Things

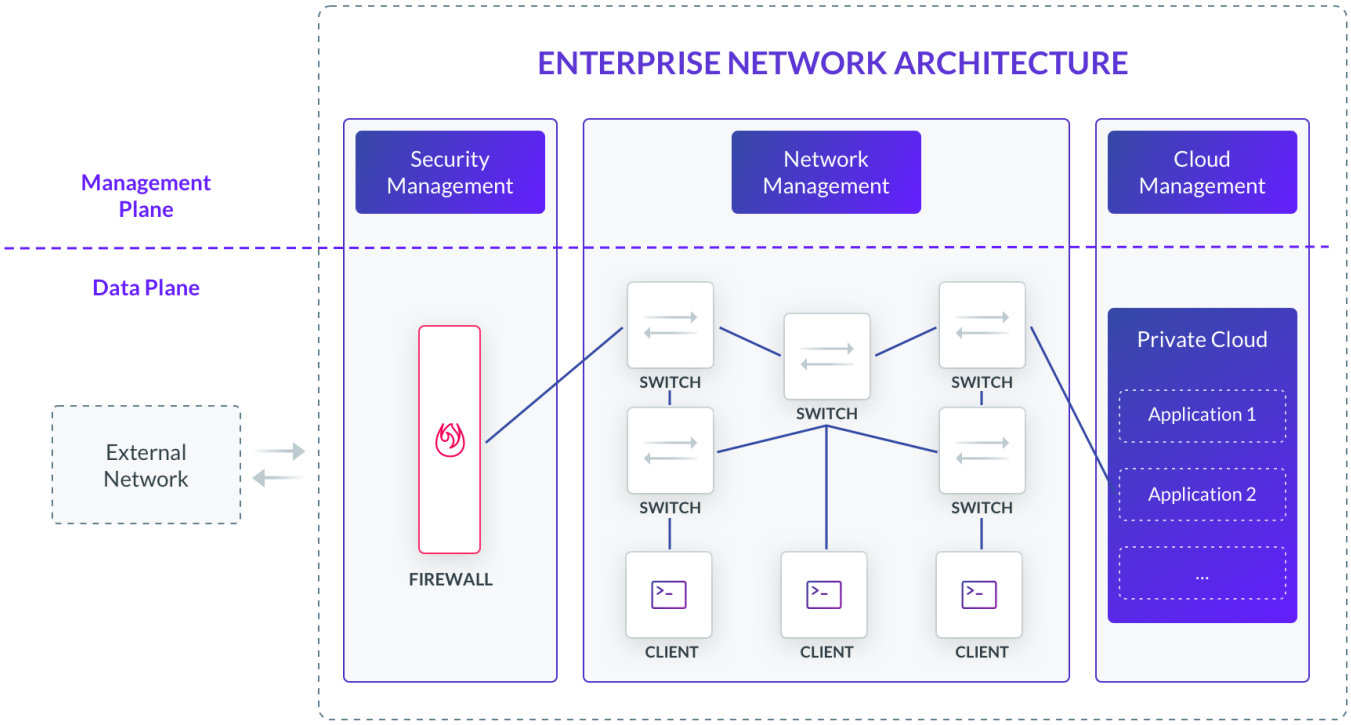
IoT represents an archetypal of the compounding of technologies. The convergence of sensors, edge computing, 5G, and machine learning, have produced a low-fidelity image of a future that looks unimaginably different. Already, it is incredible that the "Smart Cities" promised by Silicon Valley are starting to take their initial shapes in places like New York and San Francisco in the form of transportation and environmental goals.

These examples may seem like buzzwordy puffery and far removed from the majority of day-to-day challenges that organizations face, but they represent a consistent trend that all agree with: Technology continues to grow exponentially in size and scale, producing immense complexity. Infrastructure is managed by service providers, data is distributed across the globe, applications are subscribed to and communicate via APIs, new devices connect daily to internal services, and working-from-home has become universally normalized. Adapting to modernity means the decentralization of hardware, software, and people, and the abstraction of the IT stack. As if actively resisting these two trends, organizations protect resources with policies and procedures that have evolved from a centralized state. To protect critical infrastructure against the irreparable security threats presented by the rift, organizations can modernize access governance by following four best practices:

- Base decisions on identity
- Make it easy to use
- Zero Trust for all networks
- Centralize auditing and monitoring

The Current Threat

Historically, organizations have relied on protected networks to distinguish between trusted and non-trusted agents. Conceptually, this mental model was intuitive in design as resources were located within a handful of dedicated environments, meaning tools often defined access policies based on the address origin. Anything within a registered set was safe, anything without was not until determined otherwise. As the web advanced, this fledgling topology outgrown itself and organizations adapted to fragmented infrastructure. Internal networks grew in complexity, using VPNs to connect remote subnetworks together, security groups batched access behind firewalls, bastion hosts jumped users from public to private subnets, etc.



Underpinning these tactics is the same premise as when packet-switching firewalls were implemented over three decades ago: Environments can evaluate whether access was allowed, or not. Networks could be separated, gateways introduced, and user attributes defined. But after being vetted, their presence becomes trusted and left alone. Pair this philosophy with the growing scale and complexity of company infrastructure, and challenges quickly arise.

- **Dynamic Infrastructure** - The abstraction of higher-level services from lower-level hardware has made environments fluid and ephemeral. From VMs to containers, tech stacks have become increasingly compartmentalized, allowing developers to make piece-meal changes without disrupting operations. Tack on automated scaling and redundancy for high-availability and infrastructure resembles an ever-changing landscape that network engineers must map. In a static world, IP addresses and ingress ports were reliable substitutes for identifying behavior. But in a dynamic world, these variables constantly change. Keeping pace requires firewalls, routing, ACLs, and API white lists to be constantly updated. Even with sufficient automation, network engineers are burdened with tedious manual labor, driving up operational costs and slowing down development.

- **Few Internal Protections** - The complexity of mapping traffic across elastic infrastructure produces relaxed internal measures. In an ideal world, devices and services would be isolated into their smallest possible network segments, communicating through secure gateways with uniquely identifiable information. But when schedulers can spin up a container on any node within a single cluster, it becomes easier to bundle and route blocks instead of identifying individual services. Add on the reality that system administrators build back doors when security protocols impede their work, and the task of protecting resources compels a trade-off between performance and protection.

- **Over Reliance on Secrets** - Secrets like API keys, .PEM files, SSH keys, etc. unlock critical infrastructure and must be guarded with paranoid rigor. This task has turned into a near existential threat as the number of secrets has ballooned from:

  - Cloud services like storage, databases, compute, logging, etc.
  - DevOps teams deploying to bespoke development, testing, staging, and production environments
  - Machine communications occurring through the exchange of tokens, keys, and certificates
  - Internet-connected devices requiring their own secure endpoint

  Centralized secret management tools like Hashicorp Vault and AWS Secret Manager only solve half the problem. As with VPNs, once a user is given access, their activity becomes indistinguishable from all the other events, providing minimal visibility for preventative or reactive measures. Once again, IP addresses can be used as a stand-in, but they no longer inspire the same level of confidence as in the past.

Networking is a necessary component of letting the right things in and keeping the wrong things out; ports will never stop being scanned. But the expansion and adoption of technology are outstripping the ability for networks to bear the burden of defense. Modernizing infrastructure access means identifying

where and how networks struggle to cope with and comparing alternatives means. It means questioning whether effective security hurdles are being created, or agitating hoops that developers must jump through. The following section provides a set of practices that provide a foundation for these types of decisions.

Best Practices

Base Decisions on Identity, Not Secrets

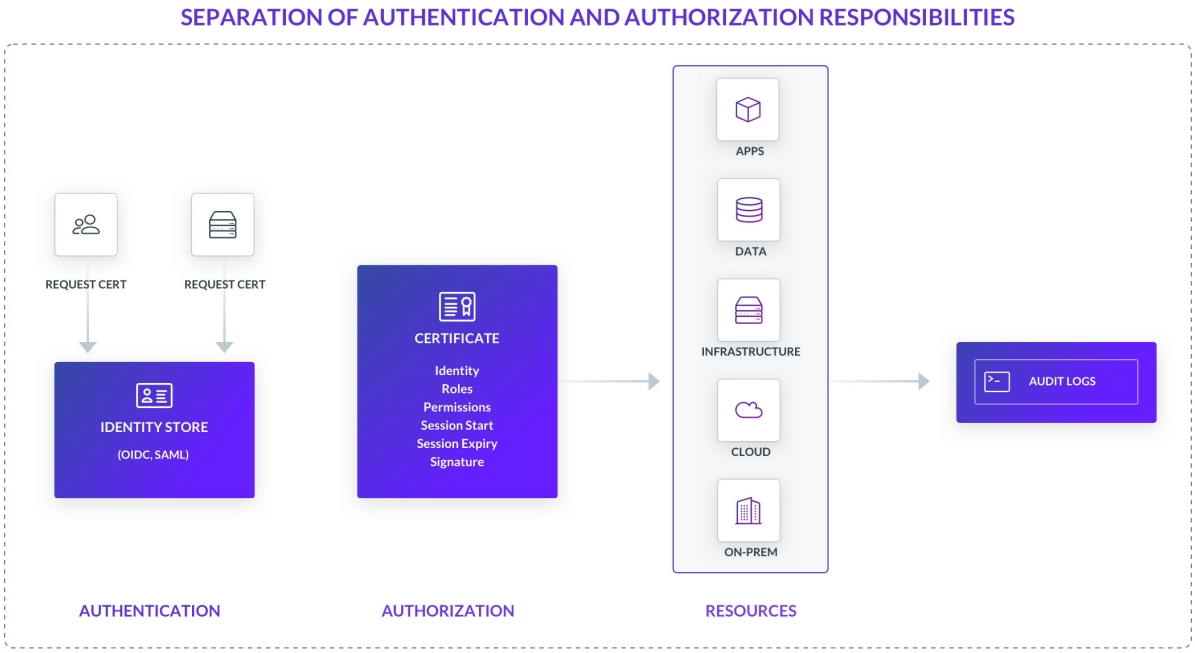
"Instead of granting access based on "what" (secret),
grant access based on "who" (identity)."

A secret informs hardware and software who or what is accessing a protected resource by proxy of possession. By rolling both authentication and authorization into one step, the client is assumed to represent the authorized party if they hold the secret. With keys being stolen, leaked, misconfigured, or lost, this is not a winning strategy. On its own, secret use does not pose a problem under the right conditions: When managed keys are provisioned to a select few with diligent tracking. But these conditions are unrealistic. Modern infrastructure generates secrets at alarming rates, leading to sprawl. Servers, virtual machines, clusters, and containers can all be ephemeral, making it difficult to track each and determine their unique set of permissions. Abstraction methods like grouping make it easier to identify and share secrets, but dilute the potency if one secret represents an entire group. Advanced protections like revocation, rotation, and MFA only add complexity to presenting secrets, not authentication.

The solution is to separate the process of authentication from authorization. Separate the process of verifying the secret holder from informing which policy is implemented. Doing so requires a single source of truth for all identity information, both humans and machines, that can reliably be queried anytime a client requests access. By outsourcing the responsibility of authentication to Okta, Active Directory, Keycloak, or others, systems do not have to issue and check secrets for each read and write request a universal identity store also affords fixed attributes for humans and clients that can be stacked to enforce authorization criteria. Identity information like group, role, and location can inform access policies at authentication. There is a difference between sharing an `admin.pem` key amongst a team of sysadmins and granting access based on Sue's grouping as a sysadmin within an AD. Secrets can be stolen, identity cannot.

It will be tempting to put engineers on their own islands where the rules apply differently to them. The logic goes that identity-based access will affect the engineer's workflow more than someone in sales, so

engineers should still be able to play fast and loose in order to meet development deadlines. But a stolen secret for production servers can be much more damaging than a stolen secret for Salesforce. This leads nicely into the next best practice.



Make it Easy to Use

"Complicated and inconvenient infrastructure access creates a less secure state."

Developers are expected to move at breakneck speeds. Effective DevOps pipelines mean pushing dozens of updates in a single day and managing clusters of constantly changing environments. When an update urgently needs to make it to production, waiting for some time-consuming redundancy measure will inevitably lead to a workaround and an unaccounted SSH key. Elaborate access processes lead to secrets hardcoded into applications, stored in plaintext in config files, or shared in private Github repos. As frustrating as it is to see developers build back doors, it signals a need to revisit design choices.

The gradual emergence of new tools and techniques marketed as DevSecOps calls for "dev-first security." This means sharing the responsibility of security between cooperative teams. By embedding automated controls throughout the development life cycle instead of existing as its own activity, user behavior is modified by complimenting it. Though this concept now has a product label, it is not new. Experienced engineers understand that good design augments behavior modification by understanding user behavior. Forcing unnatural behaviors will always result in resistance. Optimizing for ease of use may sound like sacrificing protection for production, but overdoing it will create backdoors and undermine efforts.

Zero Trust All Networks

"There are no private networks, only public."

Networks generally provide a poor form of identity, making it nearly impossible to employ a security model that restricts and isolates clients to only necessary access. Architecting infrastructure to be compatible with API driven cloud-based XaaS resources has exposed organizations to an increasing number of external endpoints over the public web. Coupled with inadequate internal protections, malicious actors can enter from one of many entry points and navigate their way through infrastructure, like the perpetrators of Target's 2014 security breach did. This axiom promotes the reality that networks can't truly be trusted. There are no private networks, only public.

Using networks as a surrogate channel to trust sets of environments must be replaced by establishing trust with the resource itself. All SaaS applications, networked devices, databases, servers, and other discrete components must be cataloged and isolated from one another. Merely existing within the same network is insufficient. Resources and clients within the same VPC/LAN cannot trust one another. VPNs also fall into this category as they establish secure connections by virtue of being their own bespoke private network. By building protections around resources instead of the environments in which they exist, lateral movement is restricted, minimizing the attack surface. This does not mean network security is obsolete. With the proliferation of the technologies discussed above, network security will only grow in importance. But they will play a less central role in the security model as users roam.

The stakes for encryption are much higher when taking this approach. Most organizations do not employ the same standards of encryption for internally routed data as they do with public traffic. Communicated over the web requires frightening amounts of trust. Unless organizations encrypt everything client-side with their own keys, which is unlikely for the modern IT stack, they cannot ensure that decrypted data has not been intercepted. Countering this problem means encrypting end-to-end. An often misused term, end-to-end encryption consists of:

- Encryption at Rest - Inactive data in digitized storage encrypted using strong methods like AES or RSA
- Encryption in Transit - Data transmitted between endpoints over an untrusted network, encrypted across protocols like TLS/HTTPS/FTPS/RDP
- Encryption in Use - Data being processed at endpoints encrypted with hashes and ciphers

Centralized Auditing and Monitoring

"Creating a source of truth for logging across cross-functional teams lets them operate with information symmetry and work in lockstep."

Sloppy git habits accrue bloat over time, as does logging. Each progressive abstraction of infrastructure generates events at a multiple of the previous. Server logs, VM logs, application logs, container logs, and Kubernetes logs all need to be aggregated, parsed, analyzed, and disposed of. A single server can go through hundreds of containers in a day, spinning them up and down for minutes at a time, each one requiring its own cross-section of the entire IT stack. Keeping these processes segregated while technical debt accrues will be like a splinter digging deeper into skin.

Centralizing logging consolidates various data silos to be processed in one location. By creating this single source of truth, cross-functional teams can operate lockstep with one another using the same accurate source. Meanwhile, teams that conduct themselves with information asymmetry will be riddled with runaway deployments and treacherous development pipelines. Unified logs also minimize questions about data integrity, making any insights immediately actionable for resource management, application troubleshooting, analytics, or SIEM.

For many markets, compliance with data privacy and security laws like SOC, HIPAA, PCI, and GDPR is the cost of entry. Meeting these regulatory or vendor compliance requirements would be improbable for modern organizations without centralized logging. The combination of both centralized logging and centralized identity lets organizations set role-based access controls with transparent auditing. Both are fundamental in order to pass any inspection.

Putting it All Together - Keys vs Certificates

The four practices above act as a rubric for evaluating what constitutes secure access as infrastructure continues to decentralize and abstract. For some organizations, this means buying the right tool. Google makes it quite easy users of GCP by packaging similar practices neatly into BeyondCorp. But for those that must build in-house solutions, this thought experiment proves to be a challenge.

To demonstrate how engineers can think about incorporating these practices into home-grown solutions, this case study will compare the usage of SSH keys vs. SSH certificates against a matrix of the best practices described above.

SSH Key Evaluation Matrix

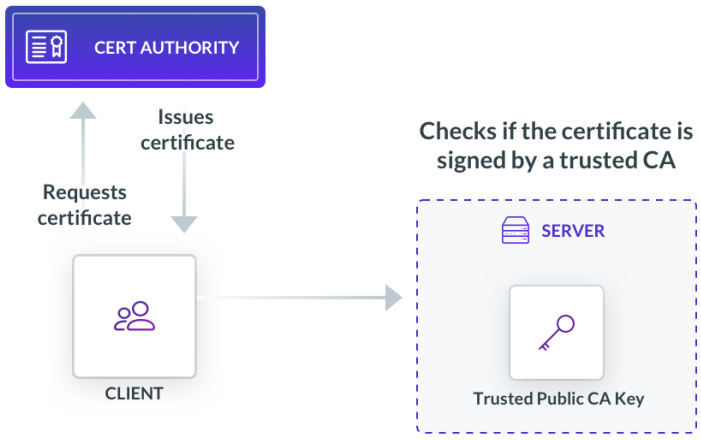
Best Practice	Performance
Base Decisions on Identity	Possession of an SSH key is not a suitable form of authentication. SSH keys are impersonal
Ease of Use	Keys grant powerful access and therefore have strict and awkward rules for how they must be used
Zero Trust	Keys are one way. Users must trust the machine they are accessing
Centralized Logging	Logs for ephemeral instances can be lost across environments. Logs often do not contain identifiable information

SSH Certificates

An SSH certificate is a public key signed by a well-known, trusted entity called a certificate authority ("CA"). A certificate authority is the ultimate grantor of trust in an organization. This means that copying keys around is no longer necessary. Users and servers simply must agree on which CA to trust and use signed certificates as a way to grant access. A more detailed primer on CAs can be found here: Certificate Authorities Explained.

SSH Certificates have a few important properties:

1. A certificate is a public key bundled with its digital signature algorithmically derived from a CA's private key. This means that only a CA can issue a certificate.
2. A certificate can be supplied with a date range for when it's considered valid. This means that certificates can automatically expire and another certificate must be issued.
3. A certificate can contain arbitrary cryptographically signed metadata, which allows a CA to encode additional instructions to how a certificate must be handled. For example, one can think of encoding various SSH options into it, like "do not allow port forwarding."



Configuring the Certificate Authority

It is generally good practice to use two different CAs: one for signing host certificates and one for signing user certificates. This separates the processes of adding hosts and adding users into two.

```
$ ssh-keygen -t rsa -b 4096 -f host_ca -C host_ca
$ ssh-keygen -t rsa -b 4096 -f user_ca -C user_ca
```

The `host_ca` and `user_ca` private keys must be protected as they are the only secrets to be issued in this scenario.

Issuing Host Certificates

Now that the host CA has been configured, it can sign host keys to generate a certificate which users can authenticate before accessing the host.

```
$ ssh-keygen -f ssh_host_rsa_key -N '' -b 4096 -t rsa

$ ls -l
-rw------- 1 ec2-user ec2-user 3247 Mar 17 14:49 ssh_host_rsa_key
-rw-r--r-- 1 ec2-user ec2-user  764 Mar 17 14:49 ssh_host_rsa_key.pub

$ ssh-keygen -s host_ca -I host.example.com -h -n host.example.com -V +52w ssh_host_rsa_key.pub
Enter passphrase: # the passphrase used for the host CA
Signed host key ssh_host_rsa_key-cert.pub: id "host.example.com" serial 0 for host.example.com
valid from 2020-03-16T15:00:00 to 2021-03-15T15:01:37
$ ls -l
-rw------- 1 ec2-user ec2-user 3247 Mar 17 14:49 ssh_host_rsa_key
-rw-r--r-- 1 ec2-user ec2-user 2369 Mar 17 14:50 ssh_host_rsa_key-cert.pub
-rw-r--r-- 1 ec2-user ec2-user  764 Mar 17 14:49 ssh_host_rsa_key.pub
```

Explaining the flags used:

- `-s host_ca` specifies the filename of the CA private key that should be used for signing.
- `-I host.example.com` defines the certificate's identity. It is an alphanumeric string that can identify the server and will be visible in SSH logs. The value here is the server's hostname. Setting this value as part of the certificate not only allows the host to be identified, but can also be used to revoke a certificate in the future if needed.
- `-h` specifies that this certificate will be a host certificate rather than a user certificate.
- `-n host.example.com` specifies a comma-separated list of principals that the certificate will be valid for authenticating. For host certificates, this is the hostname used to connect to the server
- `-V +52w` specifies the validity period of the certificate, in this case 52 weeks (one year).

Certificates are valid forever by default - expiry periods for host certificates are highly recommended to encourage the adoption of a process for rotating and replacing certificates when needed.

Issuing User Certificates

Similarly, the user CA signs certificates for hosts to authenticate users.

```
$ ssh-keygen -f user-key -b 4096 -t rsa

$ ls -l
-rw-r--r--. 1 gus gus  737 Mar 19 16:33 user-key.pub
-rw-------. 1 gus gus 3369 Mar 19 16:33 user-key
$ ssh-keygen -s user_ca -I gus@gravitational.com -n ec2-user,gus -V +1d user-key.pub
Enter passphrase: # the passphrase used for the user CA
Signed user key user-key-cert.pub: id "gus@gravitational.com" serial 0 for ec2-us-
er,gus valid from 2020-03-19T16:33:00 to 2020-03-20T16:34:54

$ ls -l
-rw-------. 1 gus gus 3369 Mar 19 16:33 user-key
-rw-r--r--. 1 gus gus 2534 Mar 19 16:34 user-key-cert.pub
-rw-r--r--. 1 gus gus  737 Mar 19 16:33 user-key.pub
```

Explaining the flags used:

- `-s user_ca` specifies the filename of the CA private key that should be used for signing.
- `-I gus@gravitational.com` specifies the certificate's identity, an alphanumeric string that will be visible in SSH logs when the user certificate is presented. Ideally this would be some unique identifier like an email address or internal username. This value can also be used to revoke a certificate in future if needed.
- `-n ec2-user,gus` specifies a comma-separated list of principals that the certificate will be valid for authenticating. In this case, this certificate gives *nix users, `ec2-user` and `gus`, to log in.
- `-V +1d` specifies the validity period of the certificate. In this case `+1d` means 1 day. Certificates are valid forever by default, so using an expiry period is a good way to limit access appropriately and ensure that certificates can't be used for access perpetually.

Inspecting the generated certificate reveals even more information, notably extensions:

```
$ ssh-keygen -L -f user-key-cert.pub
user-key-cert.pub:
        Type: ssh-rsa-cert-v01@openssh.com user certificate
```

```
        Public key: RSA-CERT SHA256:egWNu5cUZaqwm76zoyTtktac2jxKktj30Oi/ydrOqZ8
        Signing CA: RSA SHA256:tltbnMalWg+skhm+VlGLd2xHiVPozyuOPl34WypdEO0
        Key ID: "gus@gravitational.com"
        Serial: 0
        Valid: from 2020-03-19T16:33:00 to 2020-03-20T16:34:54
        Principals:
                ec2-user
                gus
        Critical Options: (none)
        Extensions:
                permit-X11-forwarding
                permit-agent-forwarding
                permit-port-forwarding
                permit-pty
                permit-user-rc
```

Logging

Certificates provide more information than static SSH keys, all of which is reflected in the logs for further visibility. Such instances include:

Certificates being used for authentication

```
sshd[14543]: Accepted publickey for ec2-user from 1.2.3.4 port 53734 ssh2: RSA-CERT ID
gus@gravitational.com (serial 0) CA RSA SHA256:tltbnMalWg+skhm+VlGLd2xHiVPozyuOPl34WypdEO0
```

Principles attempting logins that are not authorized

```
sshd[14612]: error: key_cert_check_authority: invalid certificate
sshd[14612]: error: Certificate invalid: name is not a listed principal
```

Expired certificates

```
sshd[14240]: error: key_cert_check_authority: invalid certificate
sshd[14240]: error: Certificate invalid: expired
```

SSH Certificate Evaluation Matrix

Best Practice	Performance
Base Decisions on Identity	Certificates are issued to users and hosts that have uniquely identifiable information. These identities can belong to groups with special permissions.
Ease of Use	Requesting certificates are an automated process and can be integrated into DevOps tools like Jenkins. Encoded expiry and revocation closes the loop.
Zero Trust	Both users and hosts must be authenticated by the CA endpoint. This removes reliance on private network connections like VPNs.
Centralized Logging	All requests go through CAs producing a central bottleneck to collect logging information. Identity provides enrichment.

Summing Up

Virtually all companies use SSH, which makes it a perfect example to apply secure access best practices. Variants of SSH key management systems are typically employed, but miss the mark. They can get complicated, and complexity of SSH key management will increase exposure to errors in configuration. SSH certificates fare better on the evaluation matrix. This is because:

- SSH certificates have a designated time-to-live, meaning access credentials expire automatically. In this way, secrets are automatically disposed of, perfect for accessing elastic i infrastructure without much overhead.

- Metadata injected within the certificate enriches audit logs with information like name, email, and title, and machine ID, all pulled from the centralized identity store.

- Data within certificates can be used to implement role based access controls, prescribing permissions by role and setting allow/deny rules.

SSH certificates and CAs are just one example of how to rethink secure access protocols that have outgrown their safety measures. These same principles should be applied to applications, users, devices, network infrastructure, etc.

Conclusion

Modern infrastructure looks very different from just a decade ago. Finding the right way to keep that infrastructure safe requires looking at it with different assumptions and beliefs.

- **Base Decisions on Identity**
- **Make it Easy to Use**
- **Zero Trust for all Networks**
- **Centralize Auditing and Monitoring**

These practices are not rules for implementation, but rather provide a framework for assessing authentication and authorization procedures. They apply to applications, servers, databases, devices, or any other component that forms the infrastructure.

To learn more, visit
goteleport.com