O'REILLY®

Identity-Native Infrastructure Access Management

Preventing Breaches by Eliminating Secrets and Adopting Zero Trust

Early Release

Raw & Unedited

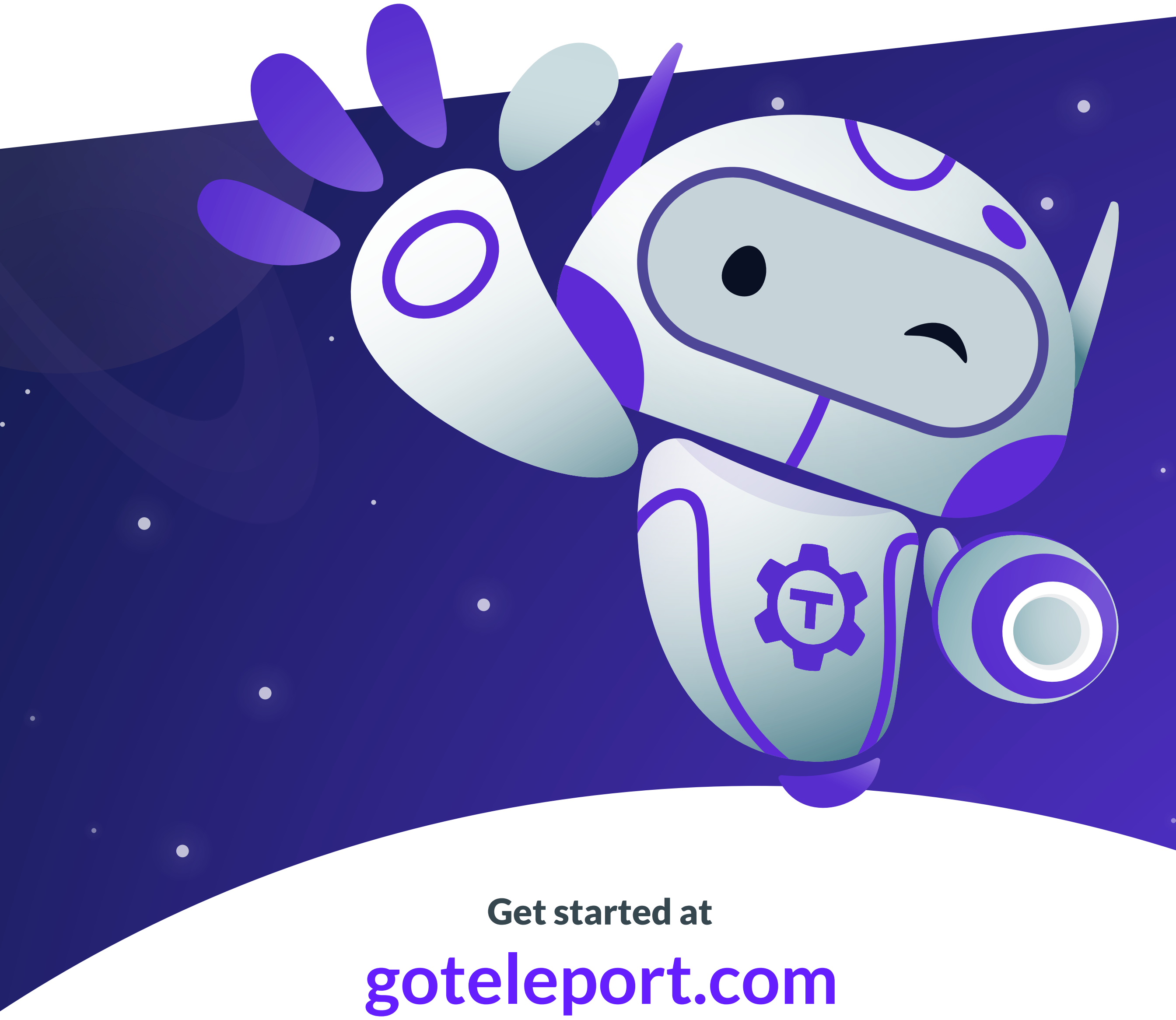
Compliments of

Teleport

Ev Kontsevoy, Sakshyam Shah & Peter Conrad

Teleport

The easiest, most secure way to access all your infrastructure.



Get started at

goteleport.com

Identity-Native Infrastructure Access Management

Preventing Breaches by Eliminating Secrets and Adopting Zero Trust

Ev Kontsevoy, Sakshyam Shah, and Peter Conrad

Beijing • Boston • Farnham • Sebastopol • Tokyo O'REILLY®

Identity-Native Infrastructure Access Management
by Ev Kontsevoy, Sakshyam Shah, and Peter Conrad

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

O'Reilly books may be purchased for educational, business, or sales promotional use. Online editions are also available for most titles (*http://oreilly.com*). For more information, contact our corporate/institutional sales department: 800-998-9938 or *corporate@oreilly.com*.

Acquisitions Editor: Jennifer Pollock
Development Editor: Jeff Bleiel
Production Editor: Gregory Hyman
Interior Designer: David Futato
Cover Designer: Karen Montgomery
Illustrator: Kate Dullea

December 2023: First Edition

Revision History for the Early Release
2022-10-05: First Release

See *http://oreilly.com/catalog/errata.csp?isbn=9781098131890* for release details.

978-1-098-13189-0

Table of Contents

CHAPTER 1

Introduction: The Pillars of Access

A Note for Early Release Readers

With Early Release ebooks, you get books in their earliest form—the author's raw and unedited content as they write—so you can take advantage of these technologies long before the official release of these titles.

This will be the first chapter of the final book.

If you have comments about how we might improve the content and/or examples in this book, or if you notice missing material within this chapter, please reach out to the authors at book@goteleport.com.

Computing infrastructure has a very broad meaning, but what we mean in this book is computing resources located in cloud environments, on-premise or co-located data centers, and even IoT devices on public networks. The definition of computing resources includes hardware such as servers, networking and storage, as well as infrastructure software such as databases, message queues, Kubernetes clusters, monitoring dashboards, CI/CD, and other DevOps tooling.

Controlling access to this complex panoply has historically relied on the notion of a network perimeter and user credentials, the digital versions of what people use to control access to their homes: a door and a set of keys. A user credential, such as a password or a private key, is nothing more than a secret piece of information that unlocks a specific perimeter. All these secrets are just data—like any data, they can be lost, shared, copied, stolen, or even sold. Neither a physical key nor a digital key guarantees the identity of someone attempting access. Simply being in possession of a secret allows access.

Like most perimeter-based access control implementations, the front door lock on a house does nothing once an intruder gains access. Anyone with the single key to the house has access to all the valuables inside. Additional perimeter defenses inside the home have the same fundamental problem. Once an attacker is inside, everything is accessible.

Corporate networks that rely on secrets and perimeter defenses have the same weakness but worse. Perimeter-based security based on secrets is inadequate because:

- The keys can be stolen, lost or have been shared with someone and duplicated later. In other words, secrets are vulnerable to human error
- As infrastructure complexity increases, there can be too many entry points to protect, increasing operational overhead
- There can be many users with different access requirements for different resources
- Once the intruder manages to gain access, they can easily pivot their attack to the adjacent resources inside the perimeter

As a company grows, a secure access model based on a perimeter and secrets does not scale.

Your computing infrastructure may consist of cloud accounts with API endpoints, VMs, databases, Kubernetes clusters, monitoring dashboards, and CI/CD tools. You may even have traditional datacenters with the same resources running in them. And every single resource requires its own access. Configuring connectivity, authentication, authorization, and audit for every resource usually involves maintaining a myriad of configuration files, each with its own compliance requirements and syntax. As the number of resources grows, the complexity of managing access to these components becomes unsustainable—and the cost of a configuration error becomes enormous.

In this chapter, we begin the discussion of identity-based access by showing how every breach or attack relies on the very characteristics that traditional security models encourage, reviewing the pillars of infrastructure access. We will lay out the ways that true identity-based access solves all these challenges by eliminating the foundation that most infrastructure attacks rely upon: human error and the attacker's ability to pivot.

Most Attacks are the Same

Most infrastructure attacks follow the same "human error + pivot" pattern.

1. The attacker first gains a foothold into a company owned computing resources by exploiting *human error*.

2. Then, an attacker pivots to gain access to adjacent computing systems on the same network.

The “human error + pivot” pattern will be mentioned numerous times throughout this book.

It is important to realize that while human errors can be minimized with security training, rigorous recruiting, and other processes, they cannot be eliminated entirely. Humans will reliably be humans.

Examples of human errors include dangerous attachments in malicious email, vulnerable web applications, a laptop left behind on the subway, an API key mistakenly committed into a public Git repository, and of course the customary sticky note on a monitor with a password on it.

Notice how every human error revolves around some kind of secret. Passwords, private encryption keys, API keys, browser cookies, and session tokens are all hubs for human error. Every secret is a potential entry point for a malicious actor. Whenever a new secret is introduced into a computing environment, the probability of a breach increases. This probability may seem insignificant at first and a successful breach may be unlikely, but as organizations scale, it becomes a question of “when” and not “if”.

> Three may keep a secret, if two of them are dead.
>
> —Benjamin Franklin

While a secret such as a password is intended to prove the identity expressed by the username, it does no such thing. Secrets-based access assumes that only the authorized person can possess the secret, but we know this is not true. A secret confers all the benefits of identity on an entity without any way to perform true authentication or authorization. Anybody with possession of a secret can pretend to be someone else. In that sense, the common term “identity theft” is misleading because what actually happens is a “secret theft”. A true identity, which we’ll get to in this book, cannot be shared, copied, or stolen.

The probability of any single human making an error that leads to a compromised secret is relatively small, especially with a competent engineering team and a strong security culture. The introduction of robust processes and secret management solutions also reduce the probability of a secret leakage to an extremely low number, but they never bring it down to zero. In practice, the difference between a low number and zero is enormous when an organization operates at scale.

To use memory corruption in a server as an analogy, the probability of a bit flip is extremely low. But as your infrastructure footprint expands and the data volumes continue to grow, eventually there will be bit flips happening every minute. That’s why error correction code is mandatory at scale: it converts the probability of a bit flip from a very small number to zero.

Reliance on a secret for access is similar. The probability of a human error leaking a secret may seem small. As infrastructure and teams grow, that small probability inevitably grows. As infrastructure becomes larger and more complex, the surface area of secrets becomes enormous, and the aggregated probability of a compromised secret becomes inevitable. That is why in a modern cloud-native infrastructure the mere presence of a secret is considered a vulnerability.

It may be tempting to reduce the risk of a leaked secret by introducing more rigid processes. Draconian security procedures not only make engineers less productive, they also create incentives for bad behavior. Hard-to-use security measures make it harder for people to comply and create harmful incentives. People tend to use shorter passwords, build their own backdoors into infrastructure, keep secure sessions open for longer than needed, try to minimize the security team's involvement in decision making, and take other shortcuts. Even the people who make the most earnest attempts to follow a difficult procedure will end up making a mistake eventually.

It's not the people who are the problem. It's the secrets themselves. Secrets are just data; data is vulnerable to loss, theft and copying.

> If you put a key under the mat for the cops, a burglar can find it, too. Criminals are using every technology tool at their disposal to hack into people's accounts. If they know there's a key hidden somewhere, they won't stop until they find it.
>
> —Tim Cook, CEO of Apple

Companies such as Google, Facebook, and other so-called hyperscalers were among the first to face this reality, and have come up with a more scalable access architecture which hinges on two crucial patterns:

- No secrets
- Zero trust

This book covers these two methodologies, explaining how they scale in "real time" with the infrastructure without increasing breach probability or attack surface dimensions.

Why does this architecture scale so well? Because without secrets, there's nothing to compromise, so human error is no longer exploitable. Instead of relying on secrets, access is based on identity itself.

Zero trust means that every user, device, application, and network address is inherently untrusted. There's no perimeter because there's no "inside" where entities are trusted. In a Zero Trust access model every network connection is encrypted, every session must be authenticated, every client must be authorized and the audit log is kept for every client action.

Zero Trust greatly reduces the chance of pivot once an attacker gains control over a single machine, and reduces the "blast radius" of an attack to just the one system initially compromised. In the world of Zero Trust, there's no difference between the company managed internal or external public networks.

Simply put, a robust implementation of Zero Trust assumes that every computing resource, no matter where it is actually located, is engineered to be safely running on a public network with a public IP address.

Access

Access enables people, software, and hardware to work together securely. At its heart, access is a collection of privileges or permissions attached to a client for a period of time, allowing it to perform certain actions on a computing resource. Managing access is the foundation of security in computer infrastructure, governing the use of all hardware and software, and how information is transferred, processed, and stored.

Access management is the ability to define and enforce how any client works with any resource. When access management relies on secrets, it is effectively giving access not to a client, but to the secret itself. This makes a stolen secret very powerful.

Before we dive into identity-based access, leaving secrets behind, let's look at what access means. At its core, remote access is based on four pillars:

- Connectivity: secure communication over an untrusted network
- Authentication: proof of a client's identity
- Authorization: specifying the actions a client can perform
- Audit: a record of real-time and historical events

These four components ensure that the right client has the right kind of access to protected resources, and that it's possible for the right others to see what's going on. The next sections provide more information about how each pillar supports access management.

Connectivity

Secure connectivity is the first pillar of access. A secure connection must be established before authentication can take place. To access a protected resource securely, an entity must be able to exchange messages without fear of interception. The legacy approach to connectivity relied on perimeter security, when encryption was needed only for messages leaving the network perimeter, also known as local area network (LAN) or virtual private cloud (VPC). Anyone within the LAN or VPC was trusted. As infrastructure grows, the network becomes more complicated. Using VPNs and firewalls to stitch together perimeters to protect trusted areas is becoming more and more challenging.

But as we've seen, perimeter-based security doesn't work because it makes you vulnerable to attacker pivots. Interestingly, security is not the only reason. It is also important to note that the perimeter died a long time ago with firewalls basically becoming speed bumps as more and more external services need connections into private networks.

That means there can be no such thing as a trusted network. This is what *zero trust* means. Encryption, authentication, authorization and audit shift from the network to the application layer, as requests no longer need to be processed based on whether they're on the trusted network. The network itself becomes untrusted, meaning that communication must be end-to-end encrypted at the session level.

Thankfully, the technologies for this were invented a long time ago, and are used for secure communications across the Internet. All of us are already using them for online banking or shopping. We simply need to properly apply the same zero trust principles to our computing environments *inside* of LAN/VPC, not only on the edge of it.

Authentication

Authentication means proving identity. A person, computer, service, or other client requesting access to a protected resource must be able to prove it is who it says it is. When you see a login screen, that's evidence of one type of authentication. Authentication must be kept separate from authorization, so that an entity's permissions can be updated if its role changes. Authentication does not determine what an entity is allowed to do. Authentication only asserts identity.

Verifying passwords is a popular authentication method, but it's inadequate for proving identity. After all, password-based authentication merely indicates possession of the secret itself and does not prove the bearer's identity. Authentication must get to the heart of identity, which is a more difficult task. How do you prove the true identity of a person in the digital realm?

One attempt at proving identity is *multi-factor authentication,* which generally uses two or three different kinds of secrets to establish proof. This pattern is sometimes called "know+have+are," and often means a password (something you know), a one-time token generated by a separate device (something you have), and our biological traits (something we are). Unfortunately, common implementations of multi-factor authentication simply convert the "know+have" pair of secrets into a session token, or a browser cookie, that means it becomes just another secret—with all the problems that a secret entails.

Authentication is a hard problem, because it means translating the true identity of an entity—who a person is—into a digital form that doesn't suffer from the same weaknesses as secrets.

Authorization

Once identity is established, authorization determines which actions a user or other entity can perform. The most well-known example of authorization is the difference between read-only access and full access.

Authorization makes it easy to see why secrets-based access is inadequate without a strong tie to an identity. Your house key gives you (or anyone who possesses it) the ability to enter your home, but it is your identity that gives you *authorization* to do so. You can grant authorization to others, allowing them to perform specific actions. You might authorize someone to repair a leaky faucet, or invite a friend to dinner. Hopefully, you're granting these permissions based on identity rather than possession of a house key.

Authorization is separate from authentication, but relies on it. Without knowing who is requesting access to a resource, it's impossible to decide whether to grant access. Authorization consists of policy definition and policy enforcement: deciding who has access to which resources, and enforcing those decisions. The matrix of entities and permissions can be very large and complex, and has often been simplified by creating access groups, and categorizing resources into groups with defined permissions. This simplifies policy management, but increases the blast radius of a breach. Someone with a stolen credential gains access to a broad group of resources based on the role to which the credential is assigned.

Audit

Audit is another reason why perimeter-based access does not scale. If you rely on a network boundary to authenticate clients, and the resources on an internal network are not protected, it means that all users become merged as a single "guest" (or worse - "admin"), making the audit logs useless.

But once access shifts away from a perimeter-based approach to resource and application level, generating more detailed and granular events, it becomes even more important to have a real-time view of as well as a historical record of access. Audit typically falls under the security terminology of manageability and traceability -- with the important point being that you actually know what is going on in your environment and have control of it.

Identity-based access management provides a great deal of control over individual access privileges, but the flip side of that is the responsibility to ensure that privileges are revoked when they are no longer needed. Regular audits can help minimize the risk of privileges being assigned incorrectly, or lingering beyond when they're needed. In other words, auditing is another hedge against human error.

Having a real-time view and a historical record of access is a critical security capability. Shifting away from a perimeter-based approach to identity-based access provides a great deal of control over access, because with audits each access can be tied back to an identity at an individual level.

Security versus Convenience

Security and convenience are famously at odds with each other. When we approach our house after a grocery run, we are forced to slow down, put the bags on the porch, and reach for the keys. This is hardly convenient, especially when it rains!

The inconvenience of security is even more evident in computing environments. Quite often, there are two groups of engineers involved in making decisions about remote access. On one hand, we have software developers who need to fix bugs quickly, ship new features to customers, improve performance, and troubleshoot abnormalities—all under a tight timeline. On the other hand, there are security and compliance engineers who are primarily concerned with risk. These two groups have wildly different incentives. The former group here doesn't want to get slowed down by security since the metrics that they are measured by typically have nothing to do with security in the first place, while the latter group is more concerned with risk than the speed of change. As a result, there's often tension between developers and security engineers, which sometimes takes the form of open conflict. A trade-off needs to be found.

> Software developers don't want to get security in the way because "in most cases the security metrics that they are measured by typically have nothing to do with security in the first place."
>
> And It's not that security wants to freeze time but oftentimes security teams don't have access to security technology that would actually help dev and eng be more productive. Historically, security technology has been all about the slow down so security professionals are stuck with what they have.

Different organizations approach this differently. Smaller technology startups err on the side of productivity, because their primary risk is not a security risk but a business risk. They may still be focusing on finding the product market fit, so the speed of product iteration is more important than compliance. As they mature, the balance starts to shift towards more robust security practices and compliance enforcement, trading off some of the product development velocity in the process.

The industry shift to cloud computing has contributed to this dilemma. Engineers have more control over their infrastructure, because the infrastructure itself is now provisioned with code. Imposing hard to follow security processes creates incentives for engineers to implement their own shortcuts for getting things done, which is easier to do during infrastructure-as-code provisioning. Often, management believes they have adopted solid security measures, while in reality their engineering team devised its own ways of accessing the cloud environments. This is called a *security theater*.

Therefore, we can conclude that an infrastructure access system is only secure if the engineering team actually loves using it.

Scaling Hardware, Software, and Peopleware

The definition of infrastructure is expanding. As remote work and personal devices become part of the workplace, and diverse computing environments proliferate, the surface area of what we once thought of as infrastructure has become impossibly complex. It's no longer practical to think in terms of networks and perimeters. Think of a company like Tesla with a network of charging stations and millions of vehicles around the globe, all of them equipped with numerous CPU, storage and connectivity. What do they deploy software updates to? Their deployment target is planet Earth!

As the infrastructure expands, we need to realize that it's not homogeneous. We need to enforce different behaviors in different contexts: development, staging, test, and production, for example. In fact, we need to protect the entire software development supply chain from vulnerabilities (a. k. a. human error) to limit the blast radius in case of a breach. Managing access securely in all these environments, with so many related goals and moving parts, is immensely complex.

Infrastructure has been able to scale quickly by moving from managing physical devices to using APIs to provision virtual devices. Because everything can be defined and managed as code (a.k.a Infrastructure as a Code), it's easy to scale elastically by provisioning more of whatever resource you need. Networks are dynamic and resources are completely fungible. Everything listens on its own network socket and needs access to some number of other resources. Tracking and blocking every network socket and endpoint to prevent infiltration would be impossible.

Ultimately, the difficulty in managing infrastructure access comes from scaling all three major elements of a computing environment:

- Hardware—the physical components that make up the system, including servers, storage, personal computers, phones, and networking devices

- Software—containers, Kubernetes clusters, database, monitoring systems, internal web applications, services, and clients that communicate with each other within a VPC or across networks
- Peopleware—the human role in information technology, including software developers, DevOps and security teams

All three of these elements are growing more complex as they scale. It's common for an organization to have tens of thousands of geographically distributed servers running more and more diverse cloud computing environments that include VMs, containers, Kubernetes clusters, databases, and an army of logging and monitoring tools. This leads to access silos, across these dimensions:

- Hardware access is siloed because cloud infrastructure is accessed differently from the older environments co-located in a traditional data center.
- Software access is siloed because databases are accessed via a VPN, SSH is accessed via a series of jump hosts with private keys stored in a Vault, and CI/CD tools are accessible on a public IP address and hooked to a corporate single sign-on (SSO) system.
- Peopleware access is siloed because some teams use manually provisioned accounts with a password manager to access their systems, others use SSO, and other teams have special requirements—such as a compliance team that allows infrastructure access from an approved laptop stored in a safe.

At the same time, as all of these teams become more distributed and elastic, relying on contractors and other outside contributors, it's necessary to quickly provision, manage, and—importantly, deprovision—access.

As we automate more and more tasks, the role of a human is supposed to be decreasing with time. To make this work, software needs the ability to communicate securely and autonomously with other software to support automated processes such as CI/CD deployments, monitoring, backups, the interactions of microservices in distributed applications, and dynamic delivery of information. Traditional security methods use tokens, cookies, and other secrets tailored to the growing number of separate tools with slightly different security protocols. This not only doesn't scale, but provides no way to track and correct human errors when they lead to vulnerabilities and breaches.

In other words, the separation between humans accessing machines and machines accessing each other creates yet another access silo: humans vs machines.

The most vulnerable component in an information system is the peopleware: users, administrators, developers, and others who access the system. Every breach can be traced back to a human error somewhere. The complexity of working with so many different access protocols, with their associated secrets and multi-factor authentication procedures, leads people to take shortcuts: remaining logged in between sessions, re-using passwords, writing secrets down, and other bad behaviors, which increases the probability of an exploitable mistake. Often, well-intended new security measures increase the drag on the people who use them, leading them to cut corners even more to preserve some level of productivity.

The point is that a growing number of access silos, each with its own secrets, protocols, authentication and authorization methods, and so on, leads to an unmanageable labyrinth of vulnerabilities that tend to increase as the complexity of access starts to interfere with people's ability to be productive.

To solve the problem, it's necessary to reduce complexity, which will not only improve the user experience but improve security by making it more manageable. While we're at it, to remove the element of human error, it would be beneficial to move away from a secrets-based model of security.

Reducing the probability of human error requires more than reducing the number of secrets. It's also necessary to tame the complexity of configuring access for a vast array of different kinds of resources, breaking down silos by bringing hardware, software, and peopleware under a single, unified source of truth for access policy.

Unifying access control across humans, machines and applications reduces the need for expertise to configure connectivity, authentication, authorization and audit in all these different systems, reduces complexity overall, and makes consistent auditability possible. Reducing complexity, in turn, gets security out of the way of convenience and productivity, giving engineers fewer reasons to take shortcuts or otherwise undermine security policy.

It turns out that there is an approach that can accomplish these goals.

Identity-Based Access

The point of identity-based access is to move away from secrets because secrets are just data and data is vulnerable to human error. True identity is not data that can be downloaded. True identity is a characteristic of the physical world. You are you. The most difficult aspect of granting access based on identity is the problem of representing physical identity digitally. Secrets tied to usernames were a futile attempt to bring user identities into the digital realm.

The idea of using a centralized identity store was the industry's first honest attempt at reducing the number of secrets within an organization. Instead of each application maintaining its own list of user accounts with accompanying logins and passwords, it makes sense to consolidate all user accounts in one database, and have it somehow shared across many applications. That's pretty much how centralized identity management (IdM) systems work. They consolidate user accounts into a single location and offer a standardized API for applications to access them. As an example, when a user requests access to a web application, the application redirects the user to an IdM system like Okta or Active Directory. The IdM presents its own login to authenticate the user, then transfers a representation of the user's identity back to the user's computer as a browser cookie, and redirects the user back to the application they are trying to access. The common protocols for this transfer are SAML, OAuth2, and OpenID Connect.

It is easy to see why this approach scales better: no matter how many applications or other identity-aware resources an organization deploys, the number of secrets will stay the same. Moreover, provisioning and deprovisioning access for employees joining or leaving a team remains the same.

While it is a big step forward, it does not eliminate all secrets entirely - just greatly reduces the number of them. And let's not forget about the browser cookie, because a cookie is just another secret, this doesn't solve the underlying problem; Steal the cookie and you can become someone else.

Another practical problem with identity management systems is that they were primarily developed for web applications in mind. SAML, OAuth2 and OpenID Connect are HTTP-based protocols. Meanwhile, computing infrastructure relies on much older resource-specific protocols such as SSH, RDP, MySQL, Postgres, and others that do not work in a browser and cannot natively be integrated with an IdM system.

Therefore, the two primary challenges in implementing identity-based infrastructure access are:

- Moving away from storing identity as secret data.
- Finding a method for transferring identity which is compatible with native infrastructure protocols.

Moving identity management away from secrets means attaching to a real, physical world identity by using biometric authentication for humans and hardware security modules for machines.

The next question is how to transfer true identity into the digital realm, because an access system needs to interact with true identity somehow. The best currently available mechanism to safely transfer true identity into an access system is digital certificates.

Certificates can be issued to machines, humans, and applications. Certificates are natively supported by common infrastructure protocols. Certificates are safer than secrets because they are far less exposed to theft and misuse. A certificate can be revoked, set to expire automatically, be issued for a single use, and pinned to a specific context (intent) and network address. As you can see, stealing a certificate is nearly pointless. The certificate chain of trust back to a certificate authority leaves only a single secret to protect—the certificate authority itself—no matter the scale of the organization. In other words, this approach scales forever without compromising security.

A modern, certificate-based central identity management system holds no other secrets besides the certificate authority. True identity of clients is stored not in a database, but in the real world as identifying physical attributes of humans and machines: human biometrics, hardware security modules, and trusted platform modules. The identity transfer is carried in a certificate which can be pinned to a context and limited in time. This brings access effectively into a single plane where all the pillars of access happen uniformly based on every entity's true identity.

This approach is the foundation not only of providing stronger security and more convenient access for users, but dealing with challenges of scale and complexity. The approach rests on the two principles mentioned earlier: removing secrets, to eliminate human error, and zero trust, to make a pivot impossible if a breach occurs.

This is the approach hyperscale companies have adopted. It's not just about moving away from secrets, but moving toward digital representation of the true identities of hardware, software, and peopleware. It's not just about encrypting all connections, but designing all infrastructure components to be safe without a firewall with no respect to the perimeter.

The following chapters explain how it's done, and why it doesn't have to be painful.

CHAPTER 2

Identity

A Note for Early Release Readers

With Early Release ebooks, you get books in their earliest form—the author's raw and unedited content as they write—so you can take advantage of these technologies long before the official release of these titles.

This will be the second chapter of the final book.

If you have comments about how we might improve the content and/or examples in this book, or if you notice missing material within this chapter, please reach out to the authors at book@goteleport.com.

Traditionally, infrastructure access has never been identity-based. Instead, it has relied on data *associated with an* identity—mainly in the form of passwords, private keys, and other secrets—which can be easily compromised, leaving critical systems open to attack. This chapter explores the difference between credentials and identity, the types of identities involved in infrastructure access, and secure forms of credentials that are not vulnerable to human error and, therefore, not prone to leaking.

We begin with your identity. An *identity* is the fact of you being who you are, usually recognized by a unique set of physical attributes that distinguish you from anyone else. These attributes include your face, your fingerprint, your DNA, and other aspects of your physical self. A server, container, applications—anything that exists—has an identity, too. The identity of an infrastructure resource is defined by its physical attributes as well, as we'll cover later in this chapter.

Establishing and proving identity is a tricky problem. It quickly becomes necessary to use *credentials,* or claims about the identity of a person or other entity. Various kinds of credentials have been invented to prove identity, and they all have their deficiencies.

At some point, whether at birth or later in your life, you're issued a national identification number such as a social security number.[1] You probably also have a license to operate a motor vehicle, including a photograph of your face, a unique license number, and other information about you. From time to time, you've probably used one of these credentials to identify yourself. In fact, these two credentials are very different from each other.

Your social security number is a unique string of digits connected to you. This number was created to track your earnings for the purpose of calculating your retirement benefits. It was never intended to serve as a form of identification. When a business requests your social security number, they generally don't ask to see your social security card. This means that anyone who knows your social security number can use it as a credential. That's a big problem, and has helped drive an epidemic of "identity theft," in which one person poses as another to obtain credit cards or engage in other identity-related crime.

Your driver's license is a more secure proxy for identity. A driver's license provides better proof of identification than your social security number, because it includes your likeness, signature, address, and the expiration date. By matching the photo to your face, or comparing your signature in person to the one on the license, anyone can have some degree of confidence that you are the person identified by the driver's license. It also contains a unique ID number. Simply knowing the number is not enough; the license itself must be physically present if you want to use it in the real world—to order a drink in a restaurant in the US where the legal drinking age is 21, for example, or during a traffic stop. This is a critical aspect of identity-based access: *secure forms of identification must never be represented as mere data;* they must be tied to physical objects. Your driver's license can be stolen, but at least it can't be stolen over the Internet.

Unlike your social security number, your driver's license represents more than just identification; it also establishes your authorization to operate certain classes of motor vehicles. That's one reason your driver's license expires periodically: the state wants to make sure you should still be allowed to drive. It's easy for a police officer or other official to check your driver's license to see if it's expired or revoked, and it's easy for a judge to revoke your driving authorization if you abuse the privilege.

The difference boils down to one thing: the social security number is merely information you know, while the driver's license is a physical object which also captures who

1 Most countries have equivalents: The SIN in Canada, the CPR number in Denmark, and the kojin bangō (個人番号) in Japan, for example.

you are: your face, your signature, and other attributes that make you unique. Even so, the driver's license is imperfect. It is not you, and the photograph is not your face.

In other words, your *true identity is not data about you, but the fact of you.* Identity is an aspect of the physical world, not the world of ones and zeroes. Much of the difficulty in proving identity comes from the challenge of representing identity—who you are—without turning it into data, which is vulnerable to theft. As the next sections reveal, many of the problems of representing identity in the physical world are the same or worse in the digital realm.

Identity and Access Management

Identity and access management (IAM), also known as *access control,* is the process of ensuring that only identified, authenticated, authorized entities gain access to protected resources, systems, or services. *Identity proofing*, verifying credentials against actual identity, is the foundation of access control. The ability to establish the identity of every client is also the key to enforcing *role-based access control* (RBAC), giving clients access only as needed for the roles they perform.

Consider the problem of managing access to a database. A database account is both an *identification system* and a form of *access control.* It includes features intended to identify clients definitively, and designed to be difficult to misuse. A user management system built into a database must do three things when you request access:

- Identification: establishing who you are
- Authentication: verifying that you are who you claim to be
- Authorization: confirming your eligibility to do something

Only if you can be identified, authenticated, and authorized can you gain access to the database. Figure 2-1 shows the relationship between these three aspects of an access control system.

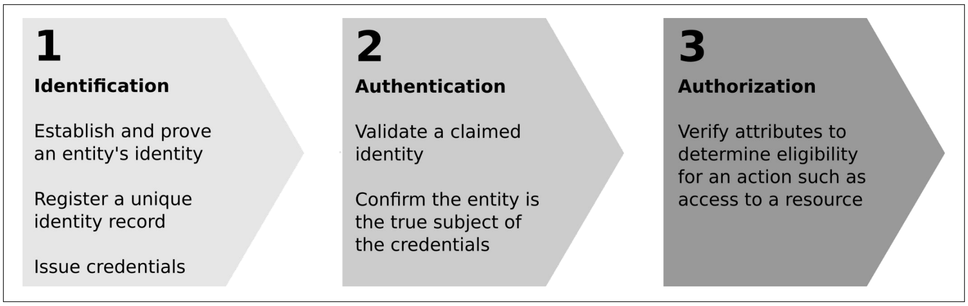

1
Identification
Establish and prove an entity's identity
Register a unique identity record
Issue credentials
2
Authentication
Validate a claimed identity
Confirm the entity is the true subject of the credentials
3
Authorization
Verify attributes to determine eligibility for an action such as access to a resource

Figure 2-1. The three parts of access control.

The first time you request a database account to be provisioned, you have to establish your identity with the database administrators. Consider the possibility of a spoofing attack at this point. Theoretically, someone else could also state your name, claiming to be you, perhaps via a falsified email message. Your name or your email address are just data about your identity, after all, not your identity itself.

To authenticate you, establishing that you are who you say you are, requires additional factors of proof. This concept, called *multi-factor authentication,* relies on several different kinds of evidence, generally including at least two of the following:

- *Something you know*, such as a username and password
- *Something you have*, a USB device such as Yubikey or a computer equipped with a unique trusted platform module (TPM).
- *Something you are*, such as a fingerprint or your face.

When you got your driver's license for the first time, you had to physically go to the DMV. You probably provided your name, date of birth (something you know) and certified documents such as a birth certificate and a utility bill (something you have), and, most important, you had to be physically present to have your photo taken (something you are). The DMV, once it was confident it had *established your identity*, was able to issue you a credential: your driver's license.

Identity and Credentials

The possession of credentials has always been the most common method of proving identity. In fact, the use of credentials for identity verification is taken for granted to the point that we tend to confuse credentials with identity. But they are different: true identity can't be stolen or spoofed. There is only one you. Credentials are merely information about you. When someone steals your driver's license or database password, they don't become you, but they might be able to pretend they are you.This difference is the root of all the vulnerabilities related to access control.

In effect, credentials are an attempt to convert real identity, which can't be transmitted or shared, into data, which can. This is useful, because it makes it possible to share trust, but it's also dangerous: like any data, credentials can be stolen or misused. When people talk about identity theft, what they really mean is the theft of credentials. Because credentials are the normal way of proving identity, credential theft creates huge problems, both in the real and digital world.

Traditional Approaches to Access

Establishing a "trust boundary" has become one of the challenging and riskiest concerns in security. Trusting valid credentials, employees, company-owned hardware devices, internal corporate networks, and managed infrastructure resources is problematic from a security perspective.

The traditional approach to establishing and using identity in the digital realm is a compromise that attempts to create a proof of identity in a form that can be transmitted. As illustrated in Figure 2-2, to interact with a computing resource, an identity is often linked to an account, essentially a database record describing a user or other entity. Each account is configured with a set of roles and permissions that provide appropriate access to resources. An account is a simplistic way of establishing identity, because it can be used by any client that can provide a valid credential. An account doesn't really require proof of identity, only possession of the credentials associated with the account.

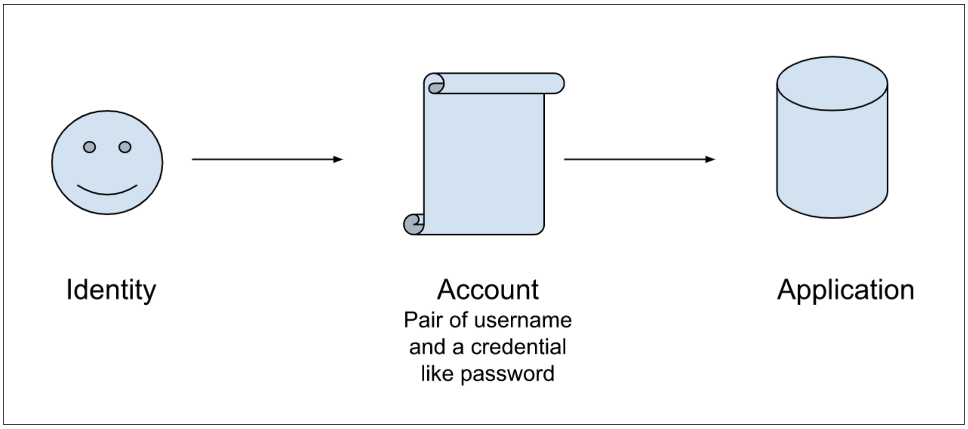


Figure 2-2. How accounts provide access to applications.

Still, accounts are a widespread way of proxying identity. In a typical infrastructure environment, a handful of users might be associated with hundreds of login accounts, each configured with multiple roles and privileges. Managing user accounts for every computing resource is time consuming. As an organization grows, provisioning and deprovisioning access becomes harder and synchronizing permissions across different types of resources approaches impossible.

To alleviate this pain, computing resources are often designed to work with accounts in groups, because it makes it possible to redefine access policy rules in batches, making an administrator's job easier. Team members sometimes share team accounts and credentials with each other for simplicity, ease of collaboration, or to save the cost of purchasing additional accounts. Even where administrators are required to manage a secure account lifecycle with periodic password rotations for administrative accounts, they may favor sharing a single administrative account with multiple users for convenience instead of creating multiple least-privilege accounts.

When accounts and credentials are shared, it becomes very challenging to track who did what and when. Account-based access, which tends toward this kind of sharing, is fundamentally flawed because it doesn't tie roles and privileges to individuals, making it impossible to audit or trace resource usage or security events. Worse, accounts often rely on secret-based credentials, which have a number of serious problems.

Secret-Based Credentials are Bad

A secret-based credential uses a protected piece of information, such as a password or key, to prove identity. If you know the secret word, you are who you say you are—or so the theory goes.

While secret-based credentials remain popular to the point of predominance, they don't prove identity. Every capability you might want for identity-based access, from building a chain of trust to verifying who really did something, is missing from secrets-based credentials. A secret is just a piece of information, so anyone who holds the secret gains access to whatever it unlocks. This makes secrets an attack point.

Every successful infrastructure attack follows the same "human error + pivot" pattern. First, an attacker exploits a human error—such as a leaked secret—to establish a foothold. Then the attacker moves laterally, trying to pivot to adjacent systems, increasing the area where the attacker can cause damage, commonly called the *blast radius*.

According to Verizon's Data Breach Investigation Report (DBIR), the most common attack patterns are compromised credentials, misused credentials, social engineering, and phishing. These patterns all have human error in common. It just so happens that most human errors that lead to breaches revolve around secrets. Passwords and other secrets are now considered a liability, because they increase the probability of human error.

Meanwhile, the number of resources, and the number or entities requesting access to them, are both growing at an astonishing rate. Servers, applications, and people are all working with more and more microservices, APIs, and data that must be protected from unauthorized access. Employees are connecting their own devices to the corporate network under bring-your-own-device (BYOD) policies, and more and more inanimate objects are joining the Internet of Things (IoT). Accordingly, the security goals for an organization should be centered around reliably identifying every entity—human, machine, or software—and granting minimal, revocable access to the resources that entity needs, during the time access is needed.

You can trust that humans will always be humans, and they can be counted on to eventually make a mistake, no matter how competent they are. It is a game of probabilities. Secret management practices, such as frequent rotation of keys and passwords, are somewhat helpful, but can't eliminate risk entirely. The more people involved, and the more secrets they rely on, the greater the probability of a human error. Think of the "surface area" of all accounts as an attack surface on which every secret represents a potential hole. At scale, in an enterprise environment, the number of secrets is gigantic. Modern enterprise infrastructure encompasses an astonishing number of entities—users, servers, APIs, applications, and the like—each with its own role, credentials, and needs for access. In these environments, every secret-based credential becomes a potential source of human error.

> "Passwords are like underwear: don't let people see it, change it very often, and you shouldn't share it with strangers."
>
> — Chris Pirillo

One can consider encryption of secrets to be a possible solution, but encryption simply shifts the attack vector from the secret being encrypted to another secret: the decryption key. This is why large scale internet companies have stopped relying on secrets entirely. Secrets simply do not scale.

Shared Secrets are Worse

In the real world, identity can't be shared—it's tied to a single entity. But an account, especially when it is configured with a secret-based credential, can be easily shared. This is a huge problem.

Even if no credentials are stolen or compromised, secret-based credentials can be innocently misused. It is a common, albeit insecure, practice to share credentials among team members. For example, the small number of people who have administrative access to a system might all use the same admin username and password. This is quite convenient! As long as everyone on the team remains trustworthy, authorized, and in the same role, Nothing Can Go Wrong™.

This pattern is even more common among service accounts and applications, when dozens of microservices may be sharing the same database username and a password injected into their code.

Unfortunately, shared secrets have some bad side effects:

- **Meaningful audit becomes impossible.** Shared secrets make it impossible to track or audit the actions of individual entities. This makes everything more difficult to manage, creates compliance headaches, and makes it much more difficult to investigate and solve problems. Because you can't identify individual users who have logged on, you don't know who did what.
- **Provisioning and deprovisioning access becomes difficult.** If a member of the administrative team leaves the organization, it must be possible to revoke that person's access individually based on identity. Otherwise, it becomes necessary to change the username and password used in common by everyone on the

team—or worse, look the other way and hope that no former employees exploit this gaping security hole.

- **Granular authorization becomes impossible.** Security best practices require granting the minimal privileges needed for accomplishing a task. With shared secrets, you can't change or revoke one person's authorization or access without affecting the whole team. The choice is either to force everyone to use a new set of credentials, leading to a disruption for the whole team, or to grant excessive privileges to everyone.

If someone on the team leaks, loses, or abuses the credentials, it's worse: you're either stuck with a big security hole, or you have to reissue the credentials and communicate them to everyone. This goes against the principle that a password is supposed to be a *secret,* a piece of information that's intentionally not shared.

Secrets are a Vector for Human Error

Human error is always there. If no one outside your organization has access to your infrastructure, it might seem that secrets plus encryption are enough. Even without a certificate, when you use SSL to access a server, the communication is encrypted. This keeps unauthorized employees from snooping—and on your own local network, you would think that you don't need to worry about man-in-the-middle attacks. However, every secret is a potential failure point. As the number of secrets grows, the probability that human error will cause a failure approaches 100%.

For the reasons described above, the mere presence of infrastructure secrets, encrypted or not, is now considered a security hole. Passwords, private keys, API keys, session tokens, and even browser cookies are all vulnerabilities exploitable by human error.

Mapping all secrets within your organization's computing infrastructure is the first step in discovering your *attack surface area*. The end goal of identity-based access is to bring the attack surface area exposed by secrets down to zero.

An Approach to Identity-Based Credentials

A credential should be derived from identity, not the other way around. When credentials are derived from physical identity and attested by a single trustworthy authority, it makes it possible to validate identities (authenticate) and enforce access policy (authorize) for hardware, software, and people. Only then does it become possible to add and verify the context of access and allow to enforce policies specific to the roles and entitlement of each entity.

It's important to create a single source of truth: a place to define policy for all four pillars of access—connectivity, authentication, authorization, and audit. During authentication, the identities of hardware, software, and people must be available to a central access control system. Unfortunately, this approach is often not the case. While most organizations do maintain a directory of employees and an inventory of computing resources, access is often provisioned in silos, resulting in duplicated or incomplete client accounts across various environments or teams. Access silos make it difficult to enforce a coherent policy.

As a result, many organizations struggle with enforcing even the basic common sense rules such as "Developers must not access production data." If the identities of developers and machines are not stored in one place, it becomes impossible at the moment of authorization to know if a client is a developer, or if a machine is a part of a production environment, or whether a machine has any data on it. When centralized identity management is implemented, each entity's access is attached to one true identity recognizable across the entire organization. This can only happen with the establishment of a central identity store for humans and machines.

Good credentials, derived from true identity, play an important role in identity-based access. These credentials must be able to carry identity "on the wire" without making it vulnerable to human error, leakage, theft, sharing, or copying.

Establishing Trust in Identity

Because true identity can't be transferred, it's necessary to create credentials that can be trusted. One approach is to create a *chain of trust,* where every credential leads back to a reliable, authoritative source of truth. This trusted authority must keep data about every credential and the identity it links to, and must be able to revoke credentials whenever necessary.

In the process of issuing a driver's license, the DMV is the trusted authority. The chain of trust looks like this:

- Everyone agrees to trust the DMV
- You must go in person to the DMV to establish your identity for the first time
- You must provide documents that prove to the DMV who you are

Only after these conditions are met can the credential (driver's license) be issued. Figure 2-3 shows how a trusted authority is central to establishing and managing identity.

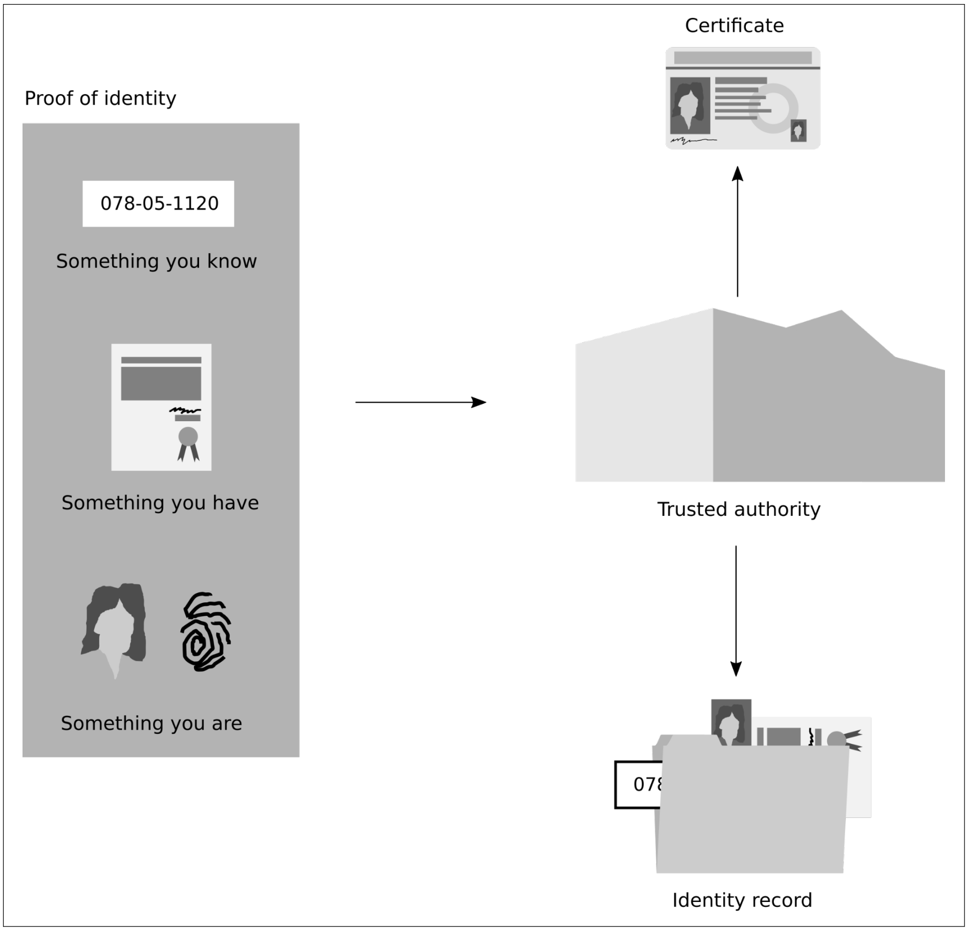


Figure 2-3. Identity proofing, recording, and certification by a trusted authority.

When the DMV issues the license, it is giving you a certificate you can use not only to authenticate your identity, but prove your authorization to drive a specific class of vehicle: motorcycles, cars and large trucks require different permissions. This authorization can be upgraded, downgraded, or revoked. The credentials provided by a database administrator have the same meaning: authentication that you are who you say you are, plus revocable authorization to access the database in specific ways.

It's important to remember that the authentication and authorization represented are independent of the certificate itself. If you alter the information on the driver's license, it no longer matches the DMV record of your identity, and the license becomes invalid. If you abuse your database privileges, they can be revoked. The ability to grant and remove authorization is central to establishing trust.

The problem of establishing a chain of trusted identity is not limited to the physical world. Just as you need a way to trust strangers with whom you perform transactions, computing resources, applications and DevOps engineers need a way to trust each other. A digital certificate helps solve this problem, and is like a driver's license in a number of ways:

- Issued by a trusted authority
- Must be presented during authentication and authorization
- Revocable, and expires periodically
- Linked to identity

Unlike a secret-based credential, a digital certificate provides a way to attest to an individual's identity, whether that individual is a person, a server, an application, or some other entity. Certificates provide not only a mechanism for establishing identity, but mechanisms for granting and revoking specific authorizations and policies. The authority that issues a certificate is responsible for issuing credentials, monitoring to detect malicious misuse of trust, and helping to thwart such attempts.

Because certificates expire, they offer additional protection against misuse. When all else fails, a certificate that's been compromised in some way will stop working at some point. By setting aggressively short expiration dates, an organization can use certificates to protect themselves against unauthorized access. Digital certificates issued by a centralized certificate authority (CA) have another important property: the CA becomes a single source of truth for both authentication and authorization for all types of computing resources within an organization, especially when every certificate can be tied to an audit log.

Establishing trust with identity certification is the first step, but the organizations must also continuously monitor the security health of devices and networks using a contextual verification modal such as zero trust. The authorization chapter discusses these methods.

Identities in Infrastructure

Establishing true identity based on the physical properties of humans and machines is the first and most important step in the access management process. The next major challenge in modern cloud infrastructure is IAM at scale. Gone are the days when a small number of people needed access to a few computer programs running on a single system. A typical environment, which now extends to the cloud, consists of thousands of servers, each running thousands of applications which can be accessed by thousands of engineers. These machines, programs, and people must all be securely and reliably identified and given access only to the services and resources they have permission to use.

For true identity-based infrastructure access (IBIA), it must be possible for any two entities of any type to establish each other's identity and enforce the appropriate access policies without being vulnerable to identity spoofing. Applications and services must be restricted to only the datasets they need, and prevented from accessing others, based on their roles and functions. A server might be in a production environment, with tight controls around data and access, or in a development sandbox where anything goes.

As explored earlier in this chapter, people's true identities reside in the real world. But what about the true identities of machines such as servers, desktops, and laptops?

When you access a Linux server for the first time with a password or key-based authentication, OpenSSH warns you of the server's identity and asks if you're willing to proceed, as you can see in the following example. If you can't—or merely don't—verify this key, you have no way of knowing whether the server is genuine, or has been planted by an attacker. This problem is called Trust on First Use. Without verifying the identifier, it's hard to trust the OpenSSH server you are connecting to. The importance of identity to trust can't be overstated.

```
user@example ~ % ssh root@192.168.1.1
The authenticity of host '192.168.1.1 (192.168.1.1)' can’t be established.
ED25519 key fingerprint is SHA256:bFBLXMrkd5fU4Qz+SB9u98oLRozzqIAbMuDGXGA95f8.
This key is not known by any other names
Are you sure you want to continue connecting (yes/no/[fingerprint])?
```

There are three important kinds of entities that require identification in infrastructure:

- **Hardware:** the machines that store data and provide access to it. This includes infrastructure resources like servers and network devices, as well as user devices such as workstations and mobile devices.
- **Software:** databases, CI/CD pipelines, Kubernetes clusters, web applications, or microservices that need access to databases or other microservices.
- **Peopleware:** people who need access to hardware and software.

All three types of entities can operate in many different roles. For example, a human might be a user, developer, administrator, or operations engineer. Similarly, a single server can host and serve multiple different applications. Traditionally, hardware identity has been handled at the network level, tracking every computer by the unique combination of the MAC address of its network card, an IP address, and a hostname.

Since the advent of virtual machines (VMs) and software-defined networking (SDN), a single server can now host multiple operating systems, making it harder to track the true identity of a service running inside the server. With container technology, a single OS can now host multiple different services, further masking the true physical identity of the server. The rise of containers has led to managing Infrastructure as Code (IaC). In this approach, containers and VMs can be *ephemeral*, meaning they exist transitionally as needed, and their identities don't persist.

This brings us to two different types of identity: long-lived identity and ephemeral identity.

Long-lived Identities

Long-lived identities are the type we're used to: they're rooted in the physical world, and can't be updated or copied easily. Because they're static, long-lived identities are more dependable and easier for use in access control. We can certify a long-lived identity once and use it as long as it meets certain constraints.

People's identities are long-lived and static. Our biological traits remain the same, no matter how old we grow. Similarly, hardware devices have attributes such as serial numbers and manufacturing numbers that are mostly static throughout their lives.

Even on the internet, the domain name system (DNS) assigns a static address for web applications. Then SSL/TLS certificates can be obtained from a trusted 3rd party certifying entity, asserting further trust in the domain name. It doesn't matter if there are changes to the underlying network addresses, hardware, or operating systems that host these web applications. As long as the domain properties remain intact, the domain names and the certificate remain the same. It's these static properties that give us trust while connecting to a website.

The question is, are these static identities dependable enough? Hardware and its serial numbers can be stolen, and software identities can be spoofed. Even people's apparent identities can change. For example, using plastic surgery, people can change the way they look or even update their biological traits to drop a previous identity, just like in the movies. On a more mundane level, "What happens if someone steals my fingerprint from a biometric device" is a valid question to ask as we head into a future of biometrics-based identity verification.

The other security risk in long-lived identities is that we also tend to assign long-lived credentials to these identities. By nature, we humans need some external push before we make a change. If a website doesn't require it,, the user may never update their password. It's the same with every other kind of infrastructure access. Engineers will probably never update credentials if not mandated to do so—not even their credentials for the primary server and databases that the whole business depends upon. In other words, the tendency is to rely on the worst security combination: the long-lived identity paired with long-lived secret-based credentials.

Ephemeral Identities

Ephemeral identities are by definition short-lived. Ephemeral identities are a byproduct of modern infrastructure management patterns: running cloud-native infrastructure with immutable and ephemeral resources. For these resources that don't have a physical presence, identity must be tied back to a physical, long-lived identity through a chain of trust.

All other things being equal, an ephemeral entity is more secure than a long-lived entity in some ways. If an ephemeral entity is compromised, it has the courtesy to disappear after a while, removing the threat. The DIE triad (Distributed, Immutable, and Ephemeral) is one of the best approaches modern infrastructure has to offer in terms of the next-generation security model.

A physical server, on the other hand, hangs around for a long time, and can become a long-standing entry point if it's compromised. One of the biggest security threats has always been the advanced persistent threat (APT), which can linger for many years inside a compromised network posing as a trusted identity.

But ephemeral identities come with a catch: how do we securely assign credentials to something that is short-lived? The security that a short lifetime brings to an ephemeral identity is less effective if a team ends up baking in static credentials such as shared secrets or long-lived tokens for convenience

Ultimately, ephemeral identities and their credentials should be tied back to the true identities of users or machines through a secure chain of identities and trust.

Identity-Based Access

Identity-based access does not rely on secret data, but uses the physical properties of hardware and biological attributes of humans instead. This drops the probability of human error in access management to zero. If every party in the access transaction has a way of asserting its true identity, then policy can be enforced without the need for secrets, opening up a secure way to ensure only authorized entities have appropriate access to protected resources.

In the context of IBIA, every access request made by clients or services must be accounted for and attributed to real-world identity and authorization, without identities or credentials being vulnerable to human error, theft, or spoofing. Managing identity-based access requires a better way to carry the physical world identity into the digital domain *without representing identity as data*. The combination of digital certificates, metadata, and a chain of trust is part of the answer. We'll start with the storage of identity itself.

Identity storage

To implement identity-based access to infrastructure in the modern, cloud-native and highly elastic world means definitively identifying hardware. To provide machine identity, cryptographic storage systems like hardware security modules (HSM) and Trusted Platform Modules (TPM) provide secure identity storage across a variety of computing devices such as servers, desktops, and laptops.

TPMs and HSMs are similar. Each of these technologies contains a microchip which stores a unique private key for the machine. The key is inaccessible for reading by software. Instead, HSMs and TPMs only offer signing, verification, and sometimes encryption capabilities. This makes TPMs and HSMs similar to human fingerprints. They can verify machine identity, but there is no data to be stolen: the machine key is “baked” into the hardware, and not accessible via any kind of an application programming interface (API). Some of these devices are even tamper proof, making it difficult or impossible to extract the key even if the device itself is stolen. This technology is available not just for physical hardware, but for virtual computing resources. The public cloud providers offer virtualized HSM services for them, such as NitroTPM or CloudHSM on Amazon Web Services.

The important principle that connects HSMs, TPMs, and human fingerprints is that *true identity is stored in the physical world*. None of the data making up an identity is accessible by a bash script, and none of it can be collected in a database. The obvious benefit is that true identity is never accessible for an attacker to steal or for an employee to accidentally leak.

However, organizations ultimately can’t get away from some need to store identity digitally, as data. Every company has a database of employees, their titles and roles, the laptops and other assets assigned to them, and an inventory of physical servers, Kubernetes clusters or virtual instances in their cloud environments. To manage identity and authorization securely, these databases and inventories must contain only the relationships between identities, but not the identities themselves. The representation of each entity’s identity is stored elsewhere, more securely. For example, the private key of a machine is stored on its HSM, and only the public key can be stored in a database. An engineer’s fingerprint is stored on their laptop’s TPM, but the TPM can produce a hash of a fingerprint signed by its private key, and that information can be stored in a database.

When access is granted on the basis of true identity, information about the relationships and roles of identities becomes less sensitive. This secondary information, no longer used as credentials, doesn’t grant access and is therefore less inherently secret from a security standpoint.

Identity Attestation

Identity-based access can only work if identity can be proven reliably. There must be a trusted party who can digitally attest to the identities of hardware, software, and peopleware. Practices such as device attestation and using TPMs and HSMs enable secure hardware identification. Similarly, an identity for each employee is established during onboarding, where biometric information is registered with the identity management system acting as a single source of truth for peopleware identification.

This lays the groundwork for separating identity from credentials.

Credentials at Scale

Credentials must also operate at scale, which means every credential must be pinned to physical identity and to usage context, so it cannot be reused elsewhere at a different time by a different party. This enables scale, because the probability of an error does not increase with the number of the current credentials in a computing environment. Stealing such credentials brings no benefit to an attacker.

To scale access management along with the speed of infrastructure growth, access and credential management must be automated, resource efficient, and low latency. This requires identity-based, scalable, stateless credentials. A digital certificate can work as that kind of credential.

How Digital Certificates Work as Credentials

Both long-lived and ephemeral entities need to be able to represent their identities safely in the digital realm, in a way that's traceable back to the true, long-lived identity of a user or machine based on a chain of identity and trust. Certificates provide a way to attest to identity, accompanied by metadata that ties identity to authorization and policy. Because certificates can expire as quickly as necessary, they also minimize the time window during which they can be compromised. These attributes make certificates a very attractive type of credential.

A certificate provides important features for authenticating and authorizing a client. A certificate:

- Contains unique identifying information about a client
- Has an expiration date and can be revoked
- Can prove its authenticity
- Is issued by a trusted third party (certificate authority)
- Can be pinned to context for when and where it can be used (network address, a client device, and a time window)

Certificates alleviate the probability of human errors: they are ephemeral, they're tied to a specific device, they are not valuable outside of their usage context and, therefore, are not a valuable attack target.

A certificate can also contain client permissions, whether the certificate holder is a person, a machine, an application, or something else. This eliminates access silos and makes it possible to enforce policies such as preventing developers from touching production data. The server doesn't have to know if it is part of the staging or production environment, whether it has any data on it, or whether the client is a developer or not. But if the server, the database software, and the client all have certificates with the appropriate metadata, then access can be granted or denied just by examining the certificates of the parties involved. This access can be granted for extremely short periods of time, ad hoc if necessary.

A certificate is the best technology we have to solve the issue of representing physical world identity with data. Certificates make it possible for every access request to be accounted for and attributed to real-world identity and authorization, without the client's identity or credentials being vulnerable to human error, theft, or spoofing. In addition, certificates provide several other benefits:

- **Identity attestation.** Upon identity verification, a CA can issue certificates to people, issue certificates to software using a code/application signing process, or issue certificates to hardware based on the keys in their hardware security modules.
- **Centralized identity management.** No matter which department an employee or service belongs to, the CA hierarchy allows distributing sub-CAs that can be configured specifically to a particular department/team's requirements, enabling centralized identity management.
- **Credential derived from identity.** Certificates are attested credentials signed by the CA. After initial authentication, the server can issue session credentials derived from the certificate—meaning that even the ephemeral credential is derived from the true identity of the person or the machine requesting access. Certificates cannot be shared, and are useless without the client's private key and their biometrics that are protected by hardware.
- **Easy and automatic revocation of the credential.** A certificate has an expiration date which automatically invalidates it after a set period of time. Further, the certificate infrastructure maintains a Certificate Revocation List (CRL) which can be used to invalidate a certificate prior to its expiration time.
- **Tamper proof credentials.** During the certificate signing process, the CA calculates the hash of the certificate metadata and cryptographically signs it with CA private key. This means all the properties of certificates, including the metadata

which contains client's roles and permissions, are tamper resistant because any changes would result in a different hash value.

- **Credential scalability.** Certificates are supported by all services and common protocols used in infrastructure management. The two common standards are SSH and X.509 certificates, which means the same certificate-signing process can scale to an arbitrary large number of clients and resources.The entire process can be automated and the total number of secrets present in your organization doesn't grow with the infrastructure.

How Certificates Are Created

In a typical access control scenario, a certificate is created like this:

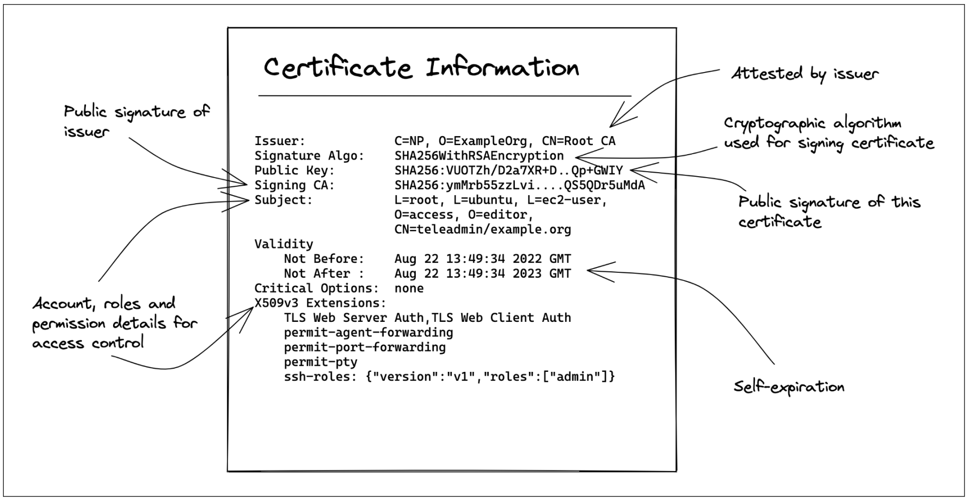
1. First, the organization sets up a central trusted entity, the Certificate Authority (CA).
2. The client must generate its own public-private key pair. The key pair can be manually generated by the client, derived from biometric input such as fingerprints, or auto-generated using cryptographic functions. Ideally, the private key should be generated from non-readable and non-accessible storage systems such as a TPM or an HSM. This keeps the private key inaccessible by anyone, including the client itself, and requires all future key signing processes to be performed within the HSM. The public key is what becomes the certificate.
3. The client establishes its identity in an onboarding process, registering with the certificate authority by supplying its signed public key. In PKI terminology, this is also known as the Certificate Signing Request (CSR). This process may require clients to perform Out-Of-Band (OOB) identity verification. This is similar to identity verification during someone's first DMV visit. When the client's identity is verified, the CA signs the client's public key with the CA's private key, establishing trust.

How Certificates Work in Access Requests

1. When the client needs access to a specific computing resource, it sends an access request to the CA.
2. The CA looks up the client in the database and verifies the authenticity of the client's public key.
3. The CA can then either sign the public key with the roles and permissions that allow access to the requested resources, or issue a new certificate for the access with a short-lived expiration time and metadata containing roles and permissions along with the public key signature of the client's identity (client's public key). This lets the access certificate easily trace back to the client's true identity.

4. The resulting certificate is sent to the client.
5. The client uses this access certificate to access required and authorized resources.

Figure 2-4 shows what an access certificate looks like.


Certificate Information
Public signature of issuer
Account, roles and permission details for access control
Attested by issuer
Cryptographic algorithm used for signing certificate
Public signature of this certificate
Self-expiration

```
Issuer:             C=NP, O=ExampleOrg, CN=Root CA
Signature Algo:     SHA256WithRSAEncryption
Public Key:         SHA256:VUOTZh/D2a7XR+D..Qp+GWIY
Signing CA:         SHA256:ymMrb55zzLvi....QS5QDr5uMdA
Subject:            L=root, L=ubuntu, L=ec2-user,
                    O=access, O=editor,
                    CN=teleadmin/example.org
Validity
    Not Before:     Aug 22 13:49:34 2022 GMT
    Not After :     Aug 22 13:49:34 2023 GMT
Critical Options:   none
X509v3 Extensions:
    TLS Web Server Auth,TLS Web Client Auth
    permit-agent-forwarding
    permit-port-forwarding
    permit-pty
    ssh-roles: {"version":"v1","roles":["admin"]}
```

Figure 2-4. Parts of an access certificate.

Reducing the Number of Secrets to One

Throughout this chapter, we have discussed the risk of secret-based credentials and how this risk has been exponentially increased in modern infrastructure management. Digital certificates address many issues related to identity verification based on secret-based credentials, but aren't entirely secretless themselves.

Certificates are based on public-key cryptography. The security of public-key cryptography depends on the secrecy of the private key. The significance of a compromised private key can be similar to a stolen password. Although certificates offer dynamic identification extensions suitable for access control, the possibility and impact of private key compromise is a valid concern.

In modern public key infrastructure (PKI), certificates are based on a chain of trust. All the end user and machine certificates are issued and signed ultimately by the root CA, as illustrated in Figure 2-5. In a typical organization, there may be thousands of certificates, but there will be only one root CA. As long as the root CA private key is protected, we can methodically contain the impact of subsequent certificate compromises.

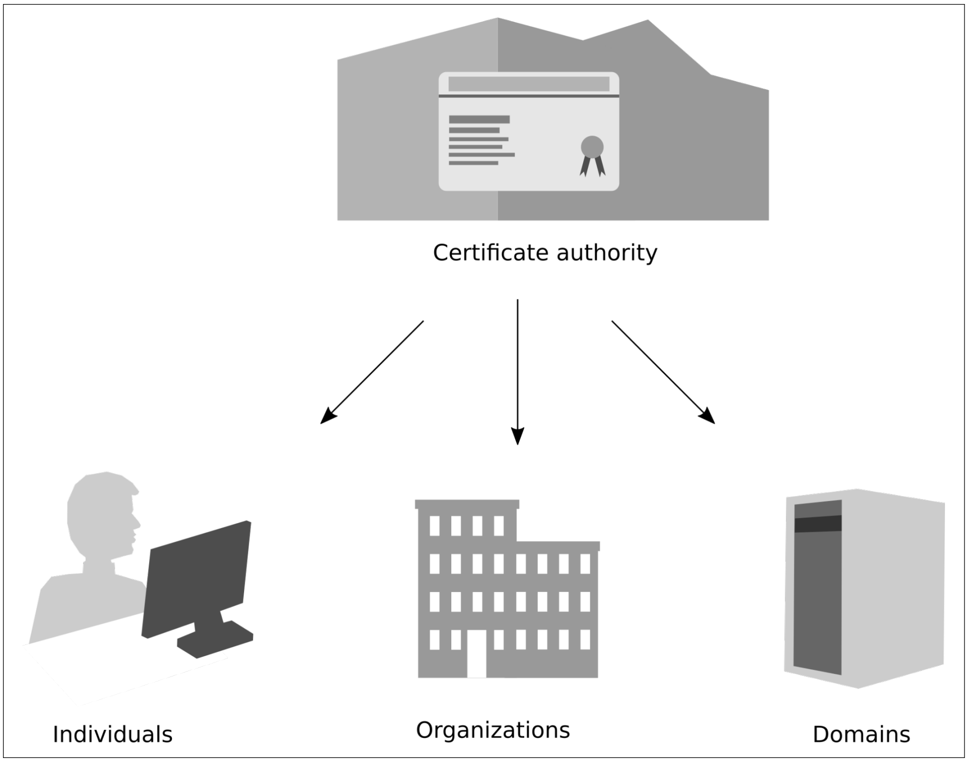


Figure 2-5. The root CA signs and issues certificates to every resource.

This is a great improvement over reliance on thousands of passwords and other secrets. Since every certificate is just a signed signature in the chain of trust, an organization that uses PKI only has a single secret to protect.

One of the values of using certificates as credentials in PKI is that it reduces the total number of secrets to one. It should go without saying that protecting a single key is easier and more reliable than protecting a thousand keys.

A Path to Identity-based Infrastructure Access

As we have learned in this chapter, identity is the core part of access control because it allows us to move away from secrets. Secrets make proper identification, proper authentication, authorization, and audit vulnerable to human error. The following sections sum up the most important tactics for moving infrastructure access towards identity-based access.

Eliminate Access Silos

Centralized identity management has seen great progress in internal applications, mostly web application access via the use of Single Sign-On (SSO) solutions. But in most organizations, there is an identity gap between corporate directory management systems like Active Directory and the infrastructure resources hosted on cloud or cloud-native environments. With cloud solutions being introduced at a rapid pace, teams operating huge enterprise IT environments struggle to keep up with an explosion of access requirements, creating access silos across different computing environments. For organizations that started their infrastructure operations in the traditional data centers, the expansion to the cloud introduces even more access silos.

The good news is that IBIA relies on open standards commonly supported by traditional and cloud-native workloads.

To make consolidating identity easier, prioritize procuring software that offers Bring Your Own Identity (BYOI) integration. Infrastructure tooling built with BYOI in mind natively supports identity management, making it easier to consolidate identity in a single source of truth for access.

Finally, consider implementing a unified access control system. Unified access control systems automate PKI management and create a common layer between all the infrastructure resources and existing corporate identities. They unify authentication, authorization, and audit for all infrastructure components. This makes it easy to "bolt-on" a centralized certificate authority and certificate-based access on top of what you already have, without making drastic changes to your current infrastructure operations. This can help break down silos without causing a painful schism in the short term.

Move to Certificates for Identity Proofing

The next iteration for identity proofing is completely passwordless. The industry standard that facilitates passwordless authentication (FIDO2 WebAuthn) is based on public-key cryptography.

Digital certificates offer security and flexibility, making them the closest digital replica of personal credentials we have in the real world. Passwordless authentication for linking the biometric material of a human to a private key on their device TPM, and issuing a short-lived certificate based on that combination, is the most secure currently available technology for identity proofing. Investing in PKI infrastructure and client devices with hardware security modules is the first step towards implementing identity-based infrastructure access.

Extend Identity-based Access to Service Accounts

In modern infrastructure operations, service accounts and bots probably access infrastructure resources more frequently than people do. Service accounts, CI/CD tools, and other infrastructure automation have been prime attack targets, as they often have privileged access to critical infrastructure resources. The separation of humans and machines represents another access silo, leading to vulnerability through management complexity. To help solve this problem, teams should tie credentials to identity for machines and bots, just as they do for users.

Summary

Secret-based long-lived identities and credentials are bad for security because they are vulnerable to human error. Security teams should be cognizant of the surface area of their infrastructure secrets and try to minimize it as much as possible. To replace secrets-based credentials, we have found that issuing certificates tied to physical properties of machines and biometric information of humans is the best form of digital credential. Certificates potentially remove all secrets from an organization's infrastructure, they are compatible with most workloads, they help eliminate access silos, and they scale well.

About the Authors

Ev Kontsevoy is cofounder and CEO of Teleport. An engineer by training, Kontsevoy launched Teleport in 2015 to provide other engineers solutions that allow them to quickly access and run any computing resource anywhere on the planet without having to worry about security and compliance issues. A serial entrepreneur, Ev was CEO and cofounder of Mailgun, which he successfully sold to Rackspace. Prior to Mailgun, Ev has had a variety of engineering roles. He holds a BS degree in Mathematics from Siberian Federal University, and has a passion for trains and vintage-film cameras.

Sakshyam Shah is a cybersecurity architect by profession and currently an engineer at Teleport. Besides cybersecurity, he loves to read and write about indie hackers, bootstrapped businesses, and early-stage venture funding.

Peter Conrad is an author, artist, and technical content strategist with experience ranging from consumer electronics and telecommunications to IoT and enterprise software.