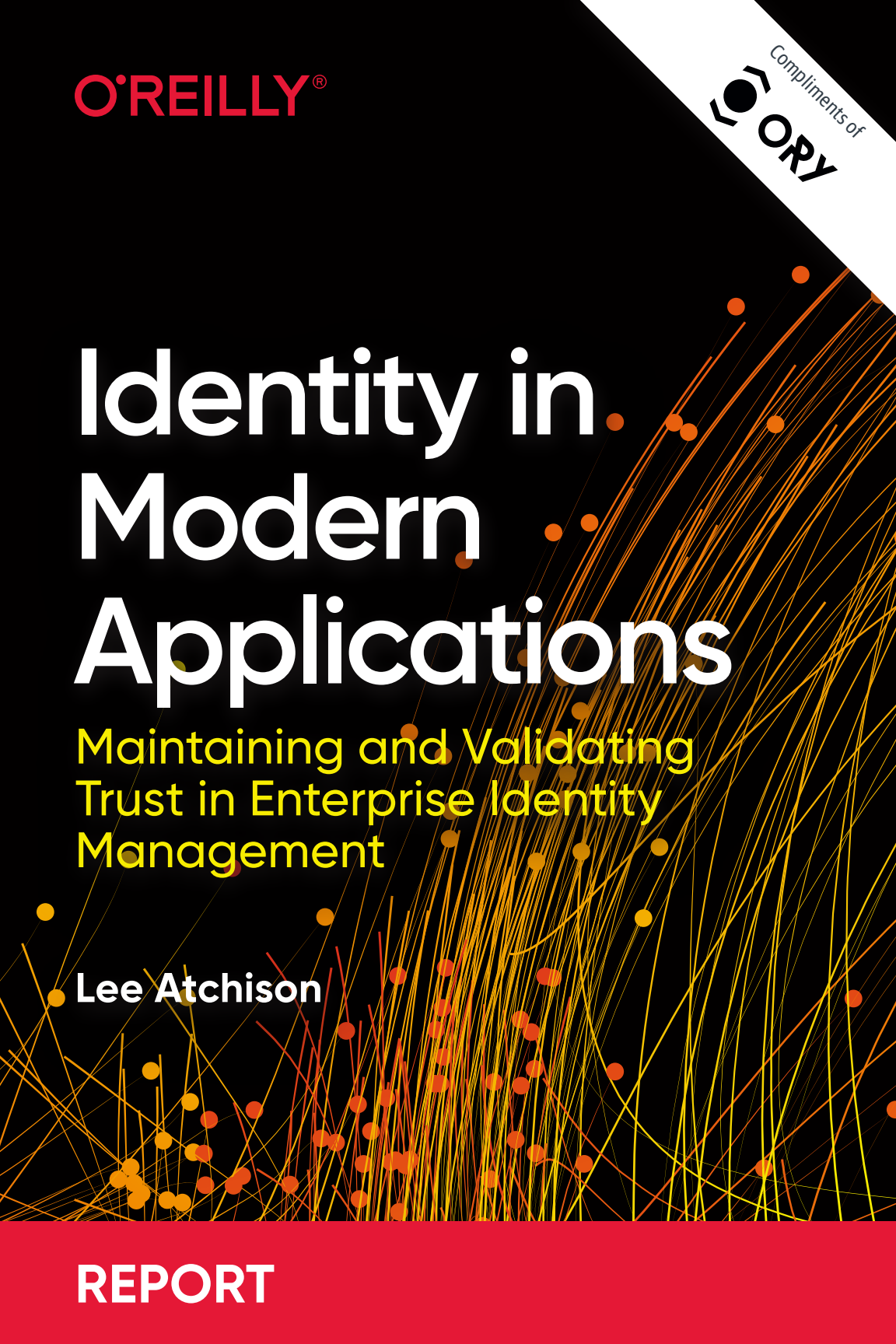
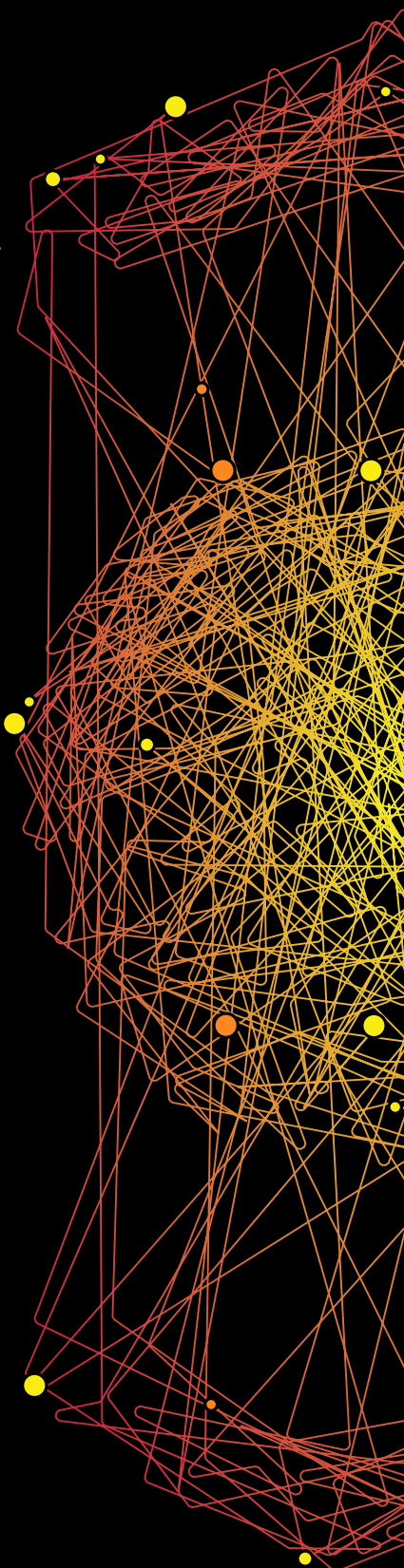

O'REILLY®
Compliments of
ORY
Identity in Modern Applications
Maintaining and Validating Trust in Enterprise Identity Management
Lee Atchison
REPORT

Unleash scalable modern auth

Speed up development, delivery and SecOps. Ory is an open source, distributed, identity and auth network for faster, safer and more robust applications.



Paying for user IDs?

Planet scale, multi region, unlimited identities.
Decrease customer acquisition costs.

Keeping customers waiting?

Unleash lightning-fast auth for instant access.
Increase sign up/in and conversion rates.

Growing through trust?

Open standards for data privacy and security.
Reduce transaction friction.

ORY

Learn more:
www.ory.sh/oreilly

Identity in Modern Applications

Maintaining and Validating Trust in Enterprise Identity Management

Lee Atchison

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

Identity in Modern Applications
by Lee Atchison

Printed in the United States of America.

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: Virginia Wilson
Production Editor: Deborah Baker
Copyeditor: Sharon Wilkey
Proofreader: Paula L. Fleming
Interior Designer: David Futato
Cover Designer: Karen Montgomery
Illustrator: Kate Dullea

July 2021: First Edition

Revision History for the First Edition
2021-07-14: First Release

978-1-098-10775-8

[LSI]

Table of Contents

Identity in Modern Applications

Maintaining, validating, and—most importantly—trusting in identity is crucial to modern computer applications. Clearly, modern identity management techniques must keep up with modern application development.

Nearly every company in existence today needs to deal with identity, and every executive, director, and manager of those companies needs to understand what identity is all about and why managing it securely is important. This report is for tech leaders who want insight into how identity works, how the techniques used to manage identity impact customers, and what future techniques are on the horizon that will meet the growing security needs of cloud-based applications.

What Is Application Identity?

In the software world, *identity* is the mapping of a person, place, or thing in a verifiable manner to a software resource. In the abstract, this is easy, but in reality, it is a complex process, especially when the identity mapping has to be verified as genuine and authentic.

Everyone deals with identity when they interact with nearly anything on the internet. A Facebook identity, an email address, and a username/password for a website are just a few examples. People deal with identities all the time.

But there is more to identity than just connecting José and Jennifer through their Instagram accounts. Everything you interact with in the real world that has a presence in the virtual world has to deal with identity.

Identity management is the term used to describe the management of large quantities of identities. For a modern application, identity management is the process of managing identities for all of the application's users, keeping usability and security in mind. For an end user, identity management is keeping track of a large number of identities for all of the applications you interact with, including usernames, passwords, and other aspects of these identities.

Example Identities

When most people think of identities, they think about identifying users—people who use a particular application. If you use Twitter, you have an identity on Twitter represented by your Twitter handle. If you use Facebook, you have an identity on Facebook. If you have an email address, you have an identity on your email service provider.

Here are some of my personal identities:

- Twitter: *@leeatchison*
- LinkedIn: *leeatchison*
- Email address: *lee###@######.com*
- Phone number: (360)262-####

An identity can be associated with a person, but identities also can be associated with anything: physical objects, virtual objects, application resources, or services. For example:

1200 Pennsylvania Avenue, Washington, DC
: The identity of the address for the White House

Seattle Mariners
: The identity of a Major League Baseball team

https://www.amazon.com
: The identity of the website for Amazon

Automobiles, airplanes, drones, robots—anything and anyone can be associated with an identity.

What Makes Up an Identity

An *identity* in a modern computer application is typically composed of authentication, authorization, and profiles. All three are required for a complete identity. Let's take a brief look at what these entail:

Authentication
: A method for a user of an identity to prove they are allowed to use that identity. When you log in to your Facebook account, you make use of your identity. First, you log in to Facebook using a username and password. This proves to Facebook that you are who you say you are. Everything you do and see on Facebook is based on this.

Authorization
: A description of permissions that a particular identity has access to within a given application or system. Once you have authenticated your identity by logging in to Facebook, you move to your favorite group and view posts that have been written in that group. Liking something you see, you create a new post in the Facebook group. The fact that your user, which you've already been authenticated as, has access to read and write posts in this particular group means you have permissions in this group. These permissions are authorization.

Profiles
: A set of information associated with an identity that can be used by the application and related services when interacting with that identity. When someone reads your Facebook post and wonders who wrote it, they see your name and avatar connected to that post. They may click the avatar to find out more information about you. The avatar is a custom attribute of your identity, and the page with information about you is part of your profile.

I'll explain these in more detail in the sections that follow.

Authentication

Authentication is the action of identifying that a user—a person, system, or resource—is who they say they are. It says nothing about whether this user has the ability to perform any actions; it just confirms who they are.

Authentication is like a passport. A passport shows you who you are, giving your picture as proof. It does not say whether you have the right to enter a specific country or not.

Authentication can be used to recognize a person who is trying to use an application, but it also can be used to recognize another external system or resource that is trying to perform a particular action with an application. Any identity, as we introduced previously, can be authenticated.

Examples of Real-World Authentication

Authentication occurs every day in the real world, outside of computers. When you walk up to someone in the street and you recognize them, you go through an implicit authentication process in your mind that validates they are the person you think they are. You might remember the way they look, the way they move, the way they smell, the way they talk.

But you can also go through an explicit authentication process to identify the person to yourself or to another person. This process might involve the following:

Physical ID
: At a government office, you might be asked for a physical ID, such as a driver's license, to prove you are who you say you are.

Shared story
: The person checking your ID may tell you a story that presumably only you and that person would know: "Hey, remember when we did that thing as kids?"

Vouch
: A third party who knows you and is known by the requestor can convince the requestor you are who you say you are: "Hey, this is Joe; remember I was telling you about him?"

Claims and Proofs

Authentication typically consists of two parts:

Claim
: A statement from a resource claiming to be a specific identity

Proof
: A set of facts or actions that validates the identity's claims

A valid authentication request occurs when a resource claims to be something, then provides proof that it is, in fact, the thing it claims to be. Here are some common ways that people can authenticate themselves to applications:

Username/password
: This is the most common and readily understandable type of authentication that occurs in a computer system. A username makes a claim about who you are, and the password is used to prove that you are this individual.

Magic link
: You share an email address or phone number, and a link or identification code is sent to that location. The address or phone number is the claim, and the ability to access the link or identification code is the proof.

Owned device
: After making a claim about who you are, the application can prove your identity by accessing a device in your physical possession. By using a device that has been previously authenticated, such as a smartphone or smart watch, an acknowledgment on that device can prove your claim.

Shared code
: A constantly changing secret code can be generated on an owned and previously authenticated device. Using that secret code can provide proof you are who you say you are.

Biometrics
: Modern devices, such as smartphones, can read fingerprints and perform facial scans to prove the identity of a human user.

Shared secret
: Any shared secret, such as an encryption key, can be used to prove an identity claim.

Single-Factor Versus Multifactor Authentication

Any single method of securing an authenticated identification, such as those listed previously, is called a *factor*. Any application that uses one of these methods for authentication is said to provide *single-factor authentication*.

The problem with single-factor authentication is its reliability. Passwords can be stolen or guessed, and a stolen or guessed password can cause an identity to be stolen. To reduce the risk of stolen or invalid authentication, a second factor of authentication is often introduced in many applications. The second factor usually involves a distinct method of authentication, as different as possible from the first method. This multiple-layer authentication is known *as multi-factor authentication*, or *MFA* for short.

MFA considerably improves the reliability of an authentication. If your password is compromised in a single-factor authentication application, for example, your identity is compromised. In an MFA application, a thief needs to steal your password *and* something else, such as to have access to your phone, to verify an authentication.

A variety of MFA techniques exist, but a commonly used technique is for the first factor to involve something you *know* (such as a password), and a second factor to involve something you *have* (such as a code or link sent to your personal telephone). This specific type of MFA is also called *two-factor authentication*.

MFA is not a silver bullet. Identities can still be stolen, but MFA is significantly safer than single-factor authentication.

Authentication Vulnerabilities

None of the current state-of-the-art authentication factors in use are foolproof. Here are some vulnerabilities to be aware of:

Passwords
: People share passwords across accounts, and they write them down. They use easy-to-guess passwords.[1] Vendors don't always encrypt users' passwords on less professional sites (so if you have a password on a compromised site, it can be tried on other sites that might use the same password). Methods designed to improve password security, such as forcing password changing and requiring complex passwords, often backfire as users balk against the complexity and retreat to easy-to-remember phrases and shared passwords that bypass the intent of the improved security. A password such as `@untSm1th` might pass simple password-complexity validators, yet still be easy for fraudsters to guess.

Text a code
: It's becoming easier for thieves to catch and receive text messages sent to a particular phone. Given only your phone number, hackers can receive a copy of all your inbound text messages, including security codes sent to your device.

Email a code or email password-recovery techniques
: Email is often used as a central repository for all your accounts. If a thief gets access to your email, that gives them dangerous access to any of your other accounts that use email for authentication. Additionally, password recovery often uses email, so a compromised email account can compromise many of your other accounts.

Biometrics
: Good biometrics (e.g., fingerprints and retinal scans) are pretty safe but can fail. Most failures involve not recognizing a valid claim, meaning the claimant has to rely on a fallback authentication mechanism. But improper recognition can also be a problem. Finding a balance between too restrictive an analysis and not restrictive enough is a challenge for most types of biometric authentication available today.

1 One of the most commonly used passwords is `123456`; another commonly used password is `password`.

None of these are foolproof, but using more than one for each given identity significantly improves your security. Most modern applications today allow you to enable MFA techniques. Enabling MFA wherever possible will go a long way toward improving your ability to keep identities secure. Using good password management techniques, such as utilizing random complex passwords unique to individual identities, will make your password itself more secure.

Identity-Recovery Vulnerabilities

Many applications also have an *identity-recovery* mechanism. This is a backdoor for recovering an identification if you forget how to access it.

Identity recovery can also give backdoor access to fraudsters. If an account has multiple security mechanisms in place to prevent fraud, yet has an easy-to-access method to *recover* a lost password, security is easy to compromise. The best password you can imagine is no defense for a website that simply asks, "What's your mother's maiden name?" before gleefully resetting your password, or sending a reset-password link via a compromised email account.

All of these are common problems that creators of modern applications deal with regularly when they build safe and successful authentication systems for their applications.

Authorization

Authorization is the process of determining whether a given user has the ability to access a specific resource—like a file, a computer, or a building—and it can be granted to any authenticated identity. Authorization is more like a driver's license than a passport. A passport (authentication) simply proves who you are. A driver's license (authorization) tells what you are allowed to do (such as drive a car but not a truck or motorcycle).

Authorization usually consists of three parts:

An actor
: This is the *identity* that wants to perform an action. It can be any authenticated identity. It could be a user, or another application or service.

A subject
: This is the *resource* that you are granting access to.

An action
: This is what the authorized actor is allowed to do with the subject. For example, if the subject is a file, the action could be *read*, *write*, or *delete*. If the subject is a building, the action could be *enter* or *turn on lights*.

A basic authorization specifies all three of these things as a specific detail, and might look something like Figure 1.

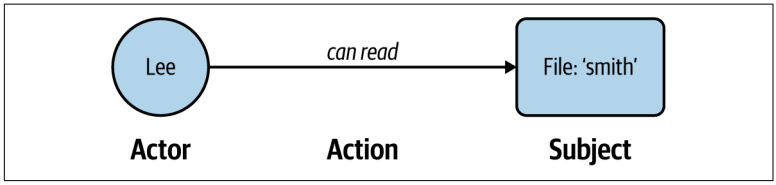


Figure 1. Simple authorization

This figure shows an authorization. It specifies that the actor/identity authenticated as *Lee* can perform the action of *read* on the subject file named *smith*. Or put simply, *Lee can read the smith file*.

Often, an authorization may have to specify multiple actions on a subject. In Figure 2, the actor *Lee* can either *read*, *write*, or *delete* the file *smith*.

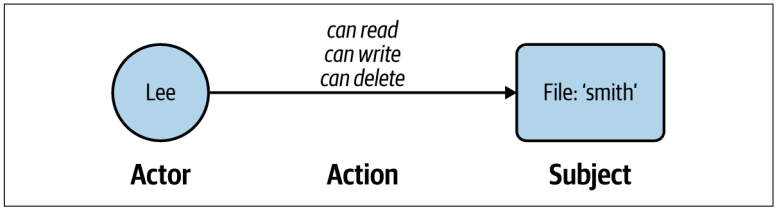


Figure 2. Multiple-actions authorization

If all possible actions are allowed, this can be generalized into *Lee can do anything with the file smith*, as shown in Figure 3.

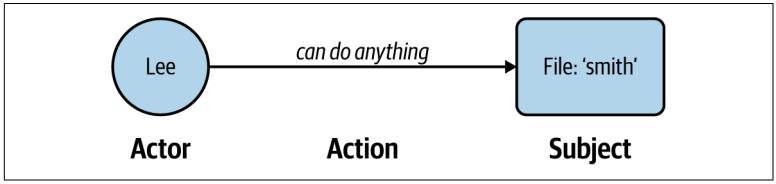


Figure 3. All-actions authorization

Subject resources can also be multiples; a specific authorization can be given to multiple subjects. In the example in Figure 4, multiple files are readable by the actor Lee.

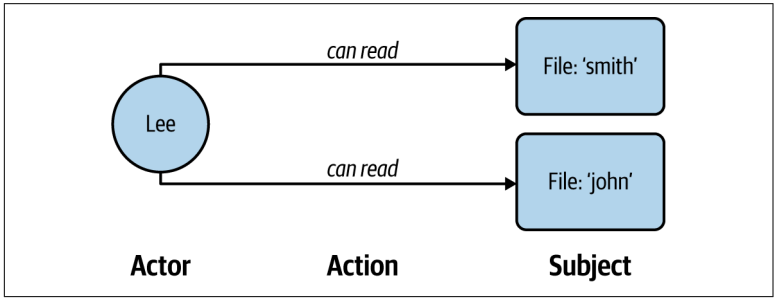


Figure 4. Multiple-subjects authorization

Authorization can also be given to groupings of resources, such as a directory containing many files. In Figure 5, the authorization indicates that Lee can read all files in the directory *TheDocs*.

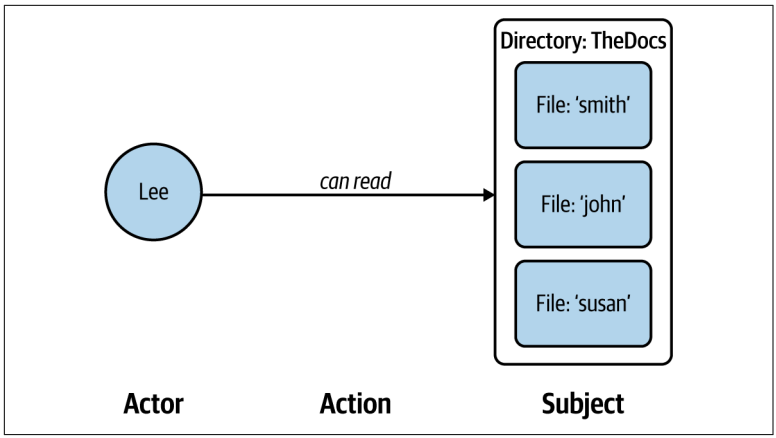


Figure 5. Groups-of-subjects authorization

Multiple actors can also be authorized, such as Lee and John in Figure 6.

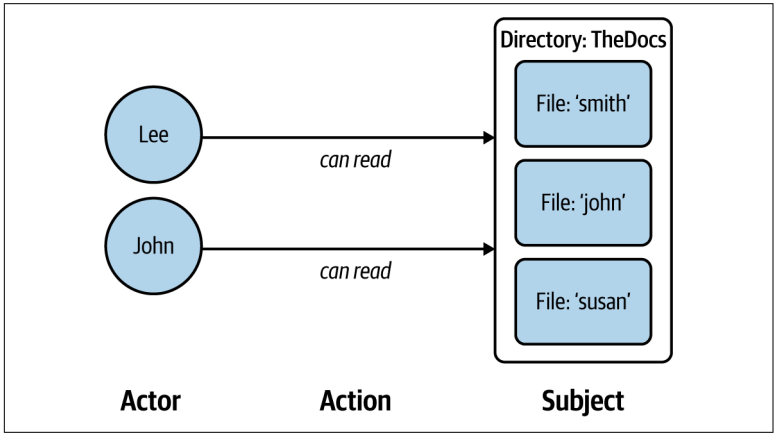


Figure 6. Multiple-actors authorization

And actors can be combined into groups of actors, such as Customers, Administrators, or Anyone, as shown in Figure 7. A group, which contains multiple actors, also acts like an actor.

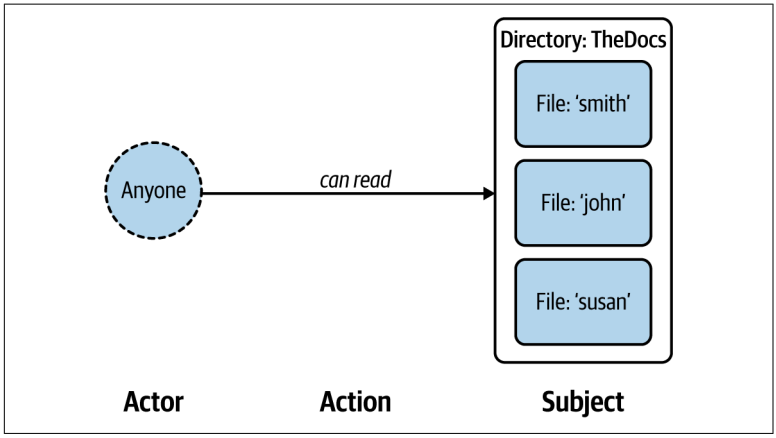


Figure 7. Everyone authorization

These authorizations can be combined and merged, providing multiple authorizations for multiple actors, actions, and subjects, creating a complex web of authorizations. The authorizations shown in Figure 8, for example, give everyone the ability to view all files, but only the *comments* file can be updated. Meanwhile, the identity named Lee acts as an administrator who can update any and all files.

This is a summary of the basics that make up authorization. This is just a representation of the types of things possible using generic authorization rules. These rules can get complex, cumbersome, and difficult to understand.

Often, an *authorization language* is used to describe these allowed authorizations. An authorization language provides a specific syntax for specifying access rules. While there are many authorization languages, I describe two common ones in the next two sections: Amazon Web Services (AWS) IAM policies and Google Zanzibar.

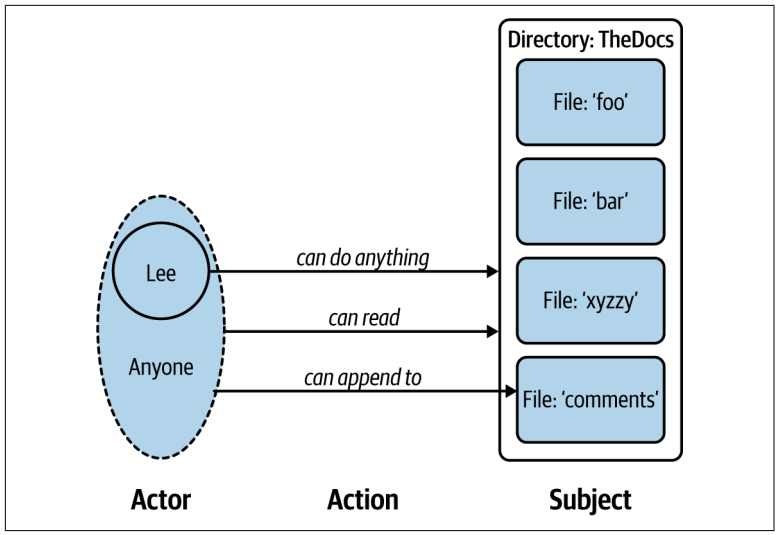


Figure 8. Complex example authorization

AWS IAM Policies

Identity and access management (*IAM*) policies typically describe the action and subject part of an authorization. The policy is then *applied* to one or more actors. In IAM terminology, the action is called an `Action`, and the subject is called a `Resource`. The authorization can describe an `Effect`, such as `Allow` an action or `Deny` an action. Here is an example policy:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": ["s3:ListBucket"],
      "Resource": ["arn:aws:s3:::test"]
    },
    {
      "Effect": "Allow",
      "Action": [
        "s3:PutObject",
        "s3:GetObject",
        "s3:DeleteObject"
      ],
      "Resource": ["arn:aws:s3:::test/*"]
    },
    {
      "Effect": "Deny",
```

```
      "Action": [
        "s3:DeleteObject"
      ],
      "Resource": ["arn:aws:s3:::test/leavethis"]
    },
  ]
}
```

This policy explicitly allows the applied actors to list the content of the Amazon Simple Storage Service (S3) bucket named `test`. Additionally, the applied actors can create, retrieve, and delete objects in that bucket. Finally, the specific object named `leavethis` cannot be deleted from the bucket.

AWS IAM policies provide detailed, granular control to resources in an AWS cloud environment. There is a lot more that these policies can do. For more information, check out the AWS IAM web page.

Google Zanzibar

Google Zanzibar provides a simple language that allows lines of tuples to describe an authorization. Each *tuple* contains an object, a relation, and a subject. This is different terminology, but it carries the same meanings introduced previously:

- The *object* is the actor.
- The *relation* is the action.
- The *subject* is the resource.

Each tuple looks like this: `object`

And here is an example: `files:my-file#access@lee`

This says that the actor `lee` has `access` rights to the file `my-file`.

By combining many of these tuples, along with more-complex names, namespaces, and identifiers, Zanzibar can create extremely specific authorization rules:

```
files:my-file#access@lee
files:my-file#write@smith
files:this-file#owner@smith
directories:MyFiles#owner@lee
```

This example says that the file *my-file* is accessible by Lee, and writable by Smith. Smith owns the file *this-file*, while Lee owns all files in the directory *MyFiles*.

There is a lot more to Zanzibar, which has multiple implementations, including the open source implementation Keto, provided by Ory. The Ory site has more information on Zanzibar and Keto, including real-world executable examples.

Profiles

A *profile* is a set of information that is associated with an individual identity. It stores information about the identity for the application to use.

A typical basic identity may consist of your real name, a description, photographs, and avatars. A basic example is what you see on a Facebook profile.

A profile might consist of saved shipping and billing addresses for ecommerce purposes. Your profile may also contain stored secrets—information that is private but required to make use of the website. This might include stored credit card numbers, Social Security number, and other confidential information.

Your identity may contain other internally managed attributes. This is information associated with an identity that is used internally by the application and is not typically of practical use to the end user. This might include things such as last login date, relationship with other identities, and types of advertisements that have been clicked.

Finally, your identity may contain an experiential record. This is a history of events, actions, and interactions with the identity in the application. This might include order purchase history, browsing history, favorites list, and product reviews created.

Challenges of Identity Management

Given the large number of users that modern applications maintain, application owners must understand how to manage these corresponding identities effectively, efficiently, and securely. Maintaining, validating, and trusting in identity is crucial to modern applications, and is growing in importance as cloud-based applications and

services become popular and interactions and connections among applications become the norm.

Globally, privacy is becoming an increasingly important aspect of identity. Laws such as the European Union's General Data Protection Regulation (GDPR) put additional restrictions on the ways companies can manage the identities—especially profiles—of their users.

Individual users have their own struggles managing identities. Every application they interact with requires a unique identity, and managing large numbers of these identities for all of the applications we interact with creates its own challenges and vulnerabilities. This increase in the number of accounts people have to deal with gives them the complex job of managing large numbers of usernames and passwords. They naturally want to simplify this complexity by using easy-to-remember phrases and repeating passwords across multiple accounts. This simplification creates an opportunity for hackers and other bad actors to steal identities.

Modern identity management techniques must keep up with modern application development and the growing sophistication of the people who interact with the applications.

Trust Systems

Together, authentication and authorization are the basis for trust. *Trust* is simply another way of expressing the relationship of an authenticated actor to an action/subject pair, as shown in Figure 9.

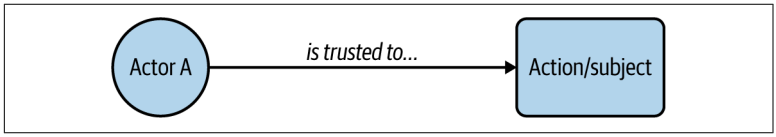


Figure 9. Trust is the relationship between an actor and an action/subject pair

In prose, here are two example statements of trust:

- *Lee is trusted to* read the file *mydata*.
- The actor *Lee is trusted to* perform the action *read* to the file *mydata*.

Describing authentication and authorization as trust is important because it allows you to talk about sharing trust between different actors. If Actor A is trusted to perform a certain action/subject, then it's possible and reasonable for Actor A to say, "I trust Actor B to do action/subject" and to have that declaration be sufficient for Actor B to now be trusted to perform the action/subject. This simple sharing of trust is demonstrated in Figure 10.

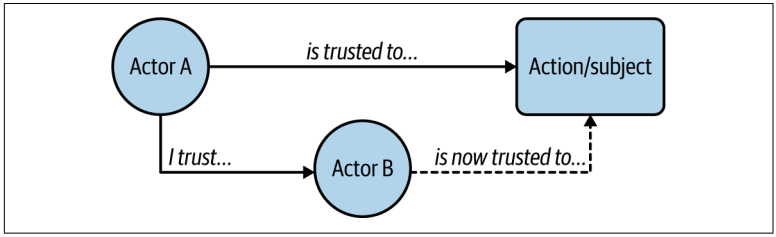


Figure 10. Actor A sharing trust with Actor B

This simple sharing of trust is called a *trust relationship*, and the system that enables this type of sharing is called a *trust system*.

This sort of trust sharing can be drawn without the action/subject being explicitly shown (Figure 11). This way of drawing the trust relationship makes it easier to follow without being distracted by the authorization details of the action/subject.

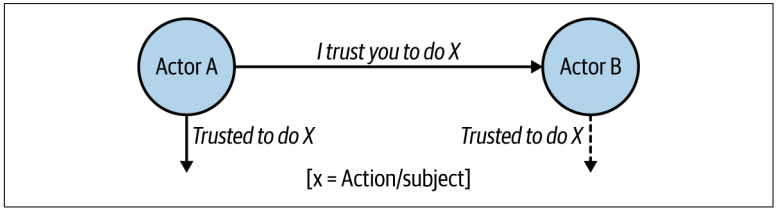


Figure 11. Simpler way to draw a trust relationship

NOTE

Just because Actor A is trusted to perform an action/subject does not necessarily mean it has the right to pass that trust on to another actor. Often, the ability to pass on trust is another authorization that has to be granted to Actor A before it can pass on trust.

Simple Trust Systems

If a trusted actor, such as Actor A in Figure 11, can pass trust on to another actor, such as Actor B, then Actor B may be able to pass trust on to a third actor, Actor C. This can allow a completely hierarchical view of trust systems. This type of relationship is called a *trust tree*. Figure 12 shows an example.

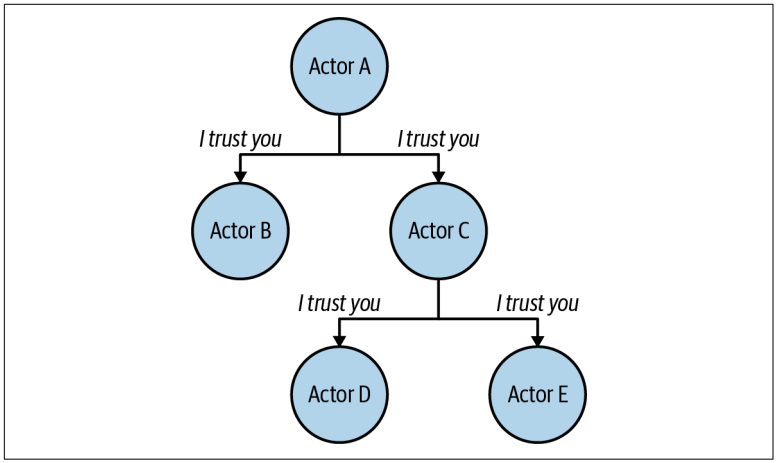


Figure 12. Trust tree

But trees aren't the only type of trust system that can be constructed. For example, Figure 13 shows a loop of trust being shared between two actors.

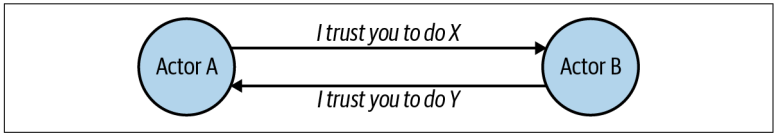


Figure 13. Trust loop

Many types of trust relationships can exist, both simple and complex. Sometimes, keeping track of who has what permissions may not be an easy task.

Why Do I Care About Trust Systems?

Trust systems allow you to scale the number of unique actors you have without needing to have centralized points of authorization management. If a new actor needs permission to perform an action on a subject, how does it get that permission? If all such requests must go to a master administrator, the total number of actors a system can have will be limited by the workload of the master administrator—the system doesn't scale. Trust systems allow the distribution of permissions to different actors to avoid centralized control.

Example: Movie Theaters

Consider a chain of movie theaters. Various decisions must be made and duties must be performed by various people in each movie theater. The owner of the chain decides which movies will be shown in the theaters. But the owner doesn't make all the decisions required to show these movies in every theater in the chain.

The owner may not decide showtimes for individual theaters, as that might vary by location. They may not decide who to hire to collect tickets and sell concessions. Instead, they delegate all the responsibility for managing each individual theater to local theater managers.

Each local theater manager is trusted to perform various actions, such as hiring/firing staff in that theater, deciding on specific showtimes for that community, and purchasing the appropriate concessions. The theater manager then takes their authority and hires assistant managers. The manager delegates to those assistant managers specific tasks related to parts of the theater. One assistant manager hires the concessions staff, another one decides on movie times and advertises them, and another hires the janitorial staff.

Lots of decisions and decision delegation is going on, all of it originating from the chain owner, but most of it occurring without that owner's knowledge or involvement. The movie theater chain trust system is scalable. This is a real-world example of a trust system in action.

In the software world, for a complex software application, often a super administrator will have full access to all capabilities of the application. They will delegate authorization to perform specific tasks to other users, who can use those authorizations to delegate other tasks to other users. In a large network, authority is driven downward, allowing the appropriate level of authorization to make decisions about lower levels of authorization. Often, user groups and group admins take on these lower-level responsibilities, while organization admins take on higher-level responsibilities.

Leveraging Trust to Improve Security

Trust systems can be leveraged to create even more-secure applications and systems. Let's talk about three specific example trust systems designed to improve the overall security of an application.

Trust but monitor

In a *trust-but-monitor system*, a responsible actor can pass on a limited amount of trust to another actor, but install capabilities for monitoring the actions of the newly trusted actor. This is also called *trust but verify*.

In Figure 14, you can see that Actor A is ultimately responsible for making sure the specified action/subject is performed safely and securely. Rather than performing the actions itself, it passes the trust to a different actor, Actor B, but installs a monitoring process. This monitoring process is designed to make sure Actor B uses the trust in a safe and appropriate manner. Actor A is informed of Actor B's use or misuse of the trust. If Actor B misuses the trust, Actor A can revoke that trust.

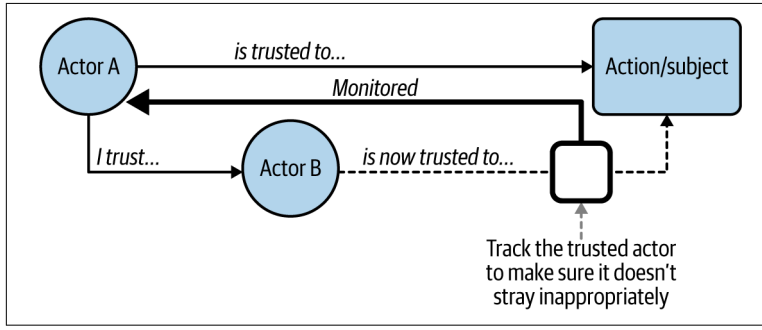


Figure 14. Trust but monitor/verify

This is a great model for training, and for actions that might be dangerous or inappropriate if left unchecked.

There are many real-world examples of trust-but-monitor systems:

- A police force that wears body cameras. The police are given trust, but the cameras provide a form of accountability.
- Certain government treaties, such as nuclear nonproliferation treaties. Verification teams often go in to verify that signature parties are following the terms of the agreement.
- Parents who let their child take their first step, with the parent watching from closely behind. The watchful parent lets the child make a mistake, but is there to help if needed to prevent serious injury.
- A call center agent taking their first solo support call, with a supervisor listening in to make sure everything happens as expected. The supervisor can interject if the new agent makes a mistake or can provide further training later.

Trust-but-monitor systems are common in software systems as well, with log files commonly being used as the monitor/verification system to check on the various actions taken by administrators on the site.

Partial trust

Often a task is sensitive enough that no single actor has permission to perform the entire task. Instead, the task requires multiple trusts, each trust provided by a distinct actor, as shown in Figure 15. This is called a *partial-trust system*.

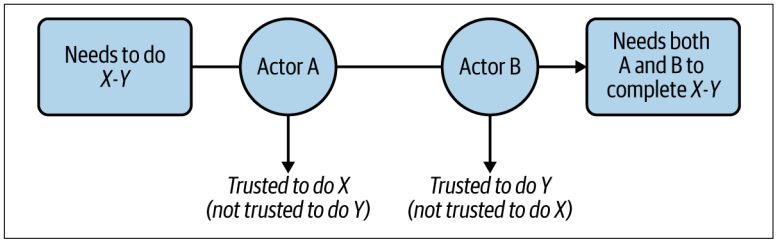


Figure 15. Partial trust involving multiple actors

In a real-world example, an employee may need to make a purchase for a company and receive a reimbursement for that purchase. This

process typically involves three partially trusted actors, each performing distinct jobs:

The employee
: This person is authorized to make the purchase by using their credit card, usually up to certain limits and conditions.

The employee's manager
: The purchase is reviewed by a manager, who can approve the reimbursement for the purchase.

The finance department
: They review the employee's purchase, along with the manager's approval, and determine whether it is safe and appropriate to pay the reimbursement. The finance department issues the reimbursement check to the employee.

All three trusted actors are required for the purchase to be paid. The employee can't cut themselves a reimbursement check, nor can the manager. Only the finance department can execute the reimbursement. But the finance department can't issue a reimbursement unless the employee provides a receipt and the manager approves the transaction. Without the support of all three actors—and the trust granted to each of them—the transaction cannot happen.

Figure 16 shows this trust system.

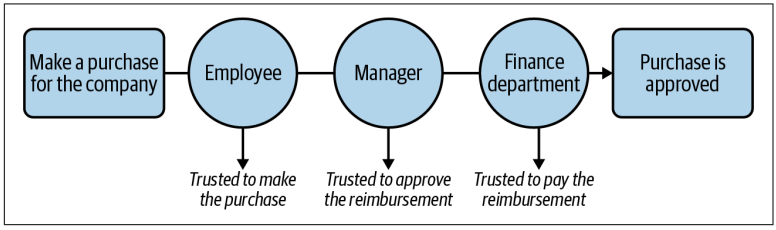


Figure 16. Employee reimbursement trust system

Another name for partial-trust systems is *checks and balances*. Many types of partial-trust systems are used in the real world. Here are a couple of examples:

Nuclear submarine

The movie *Crimson Tide* has a storyline based on a partial-trust system. It's a story about a United States nuclear submarine that is given an order to fire its missiles. The captain of the sub is preparing to fire the missiles, but the executive officer (XO) refuses to give permission. In order for the missiles to be fired, both the captain and XO must give permission.

US government branches

The US federal government is designed with checks and balances that create a partial trust system. Each of the three branches of government is trusted to perform certain types of actions: the Congress makes laws, the president executes the laws, and the judiciary enforces, interprets, and validates the legality of the laws. The government cannot function without all three branches performing their jobs, and overreach by one branch is controlled by the independence of the other two branches. This is an example of a large-scale partial-trust system maintaining a democracy.

Zero trust

Imagine an application, composed of multiple services, with the backend infrastructure running behind a corporate firewall or virtual private cloud (VPC). Figure 17 shows a simple example.

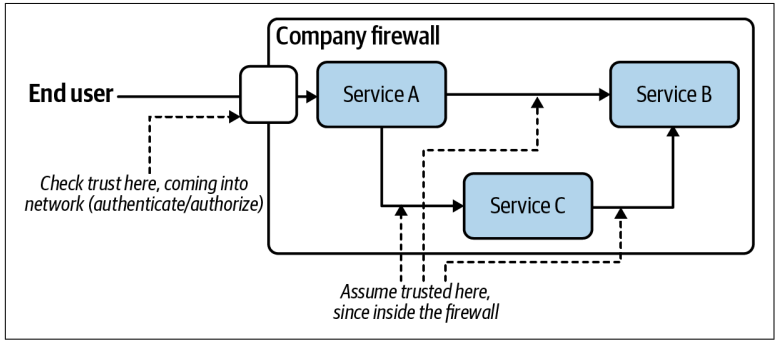


Figure 17. Typical service-based application in a private infrastructure

The end user has access to this application by going through an authentication system at the frontend. This is the trust point. The end user is authenticated and the authorization is checked, and if valid, is allowed to pass the request on to Service A. Service A has

direct and easy access to the rest of the application—namely, Services B and C.

But what happens if Service A gets compromised? Then, almost by definition, you have to assume that Service B and Service C are also compromised. In fact, everything inside the company infrastructure firewall has to be assumed to be compromised. This is because there are no security checks, no authentications, no authorizations, no trust checks at all, once you are inside the company firewall. A bad actor with access to Service A can run amok in the entire application backend.

Instead, what happens if you insert a trust check at each service connectivity point? In other words, not just for end users accessing the network, but for every internal service talking to every other internal service. This is illustrated in Figure 18: every service call is authenticated/authorized and allowed to be executed only if the trust check passes.

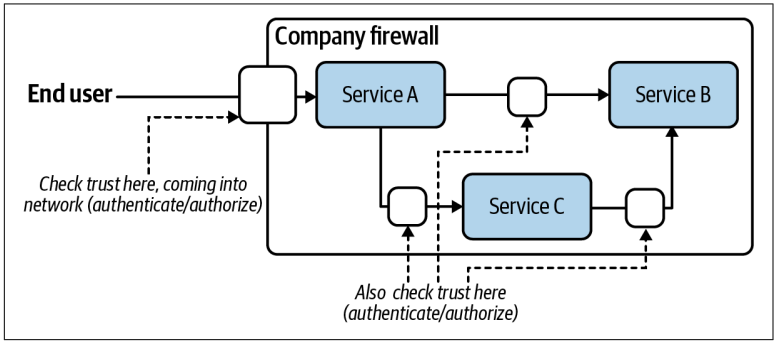


Figure 18. A zero-trust system with trust checks at every service boundary

Now, what happens if Service A gets compromised? Since someone with access to Service A still has to go through a validation check to get to the other services, the bad actor with access to Service A is stuck. They don't necessarily have access to anything else inside the network. This is because Service A doesn't have any elevated access to the rest of the backend infrastructure.

This is an example of zero trust. No service, system, component, user, employee, or operator has any permission to access any part of the infrastructure simply because they have access to another system. All components and players are essentially untrusted and have

to prove their trustworthiness at every turn. Zero trust creates a more secure backend.

An infrastructure that effectively uses zero trust typically also uses partial trust. This is because Service A has the right to perform certain operations in the infrastructure that Services B and C cannot perform, and Service B can perform some operations that Services A and C cannot perform. This is a form of partial trust.

The same partial trust extends beyond services to all aspects of the backend infrastructure, including to the operations personnel who will operate the backend. A zero-trust system typically has no *super user*—a person who has full access to everything in the system. Instead, different operations personnel have access to only the parts of the system they require to perform their jobs, and no more. Multiple independent personnel are required in order for complete access to the entire system to be available. Therefore, no single operator can inappropriately access the entire application, only the parts they are responsible for. Taken to an extreme, no single operator can destroy the running application without assistance from other operators. This provides a significantly higher level of security from internal bad actors as well as external bad actors.

The concepts of zero trust and partial trust often overlap significantly in infrastructure applications.

Picking a Trust System

Obviously, the specifics of an appropriate trust system depend on the use case and the needs for the trust system.

It would not be appropriate for a nuclear submarine to be managed using the trust-but-monitor system, enabling the captain to fire missiles anytime they want, with their actions reviewed later. That would be a dangerous environment to live in. Instead, a partial trust system is required, so the missile cannot be fired without multiple actors agreeing to the action.

A parent uses trust-but-monitor to help give freedom to their children while they are still learning. A parent has to "let go" but still keep an eye on a new toddler who is learning to walk. It would not be appropriate for the parent to require the child to get permission for every step, because the child would never learn independence.

A computer infrastructure may assume that a trust-but-monitor system is appropriate to make sure that no bad actor gets into the backend infrastructure, and that any infiltration is detected by the monitoring system. Yet, even with this system in place, additional partial-trust and zero-trust systems may improve the security and safety of the infrastructure if it does get compromised.

Only after a thorough analysis of your specific security needs can you decide which type of trust system makes sense for your application and organization. You must consider the risk of broken trust in a trust-but-monitor system against the cost, complexity, and scalability impact of a tightly held trust to ensure security.

Future of Identity Management

We are at a crossroads in time when identity is changing as follows:

Becoming more prevalent
: As we consume more online applications, more connections will be created among those applications. This means a greater need to keep track of many more identities.

Under greater attack
: More and more bad actors are engaging in the process of stealing identities and forcing their way into important applications. Identity security requires more-sophisticated approaches.

Becoming more difficult to manage
: The addition of second factors, required password complexity, password aging, rules around reusing passwords—all designed to keep our systems safe and secure—have added complexity to end users' lives and encouraged bad identity management habits, such as password sharing, reuse, and simplification.

For us to keep moving forward as a technologically enabled society, identity management must improve.

Thing You Know, Thing You Have

We talked previously about two-factor authentication (a form of MFA), which uses something you know (such as a password) and something you have (such as your cell phone) to authenticate a user. This is a solid strategy for improving software security.

However, two points need to be addressed for two-factor authentication to be truly safe and secure in the long term:

Factors must actually be secure
: As we've discussed, some factors are safer than others. Passwords often are easy to guess, and text messages are often seen as more secure than they really are.

Factors must be convenient
: Early second-factor authentication methods required carrying a hardware token or dongle with you at all times, and using that device as the "something you have" second factor. This is inconvenient and encourages people to work around the need for a second factor, as well as limits the update on second-factor adoption.

But there is hope. Improved technology in authenticator applications, such as Google Authenticator, provides the ability to manage a second factor on your cell phone rather than needing to carry a separate hardware device with you. Using such an application can allow you to provide a second factor to many applications at the same time.

Using a password manager allows you to store random and unique passwords for each application you utilize, dramatically increasing the security of your passwords. Some password managers, such as 1Password, are even building in second-factor authentication, allowing them to replace Google Authenticator and automatically and conveniently fill both your password and your second-factor authentications into applications.

Future improvements in technology will make secure MFA even easier—for example, by building two-factor authentication techniques directly into browsers and by using invisible second factors, such as encryption keys stored in the browser data for a given site.

Nonhuman Identities

As microservice architectures, cloud, and software-as-a-service (SaaS) applications become more common, safe, and secure, communication between applications and services becomes even more critical.

Often, connecting one application to another involves sharing a human user's identity and storing it in one application for use in

authenticating to the other application. This is not only unsafe and unwise, but also potentially problematic. What happens if the user assigned that identity leaves the company? Does the link between the two applications suddenly fail?

Nonhuman identities are an important aspect of the identity solution, and are critical for zero-trust and distributed partial-trust systems. Given that no human is involved in the identification process, these systems do not require less-than-optimal solutions catered to making the authentication process easy for a human to use. As such, lengthy keys, encrypted messages, and constantly rotating keys are technologically possible and reasonable, and should be used more often.

Additionally, newer technologies such as temporary identities and *identity hopping* (using multiple temporary identities that change from one request to another) are possible and dramatically improve intra-system security.

Global Identity: The Value of Network Effects

Imagine an office worker in an office building. They walk outside to the line of food carts that make their way into the neighborhood around lunchtime every day. They find one that interests them, walk up, and order their favorite meal. The vendor asks for payment, and the worker inserts a credit card into the payment terminal. It processes the payment and then asks, "Would you like me to email you a receipt?"

At this point, the office worker is a bit miffed. They typically keep receipts for lunches, but the food cart doesn't have a printer, so if they want a receipt, they have to spend a minute—a hungry minute with their lunch getting cold—entering their email address into the virtual keyboard on the payment terminal. After entering it, and hoping they didn't type it in wrong, the receipt arrives in their inbox five minutes later, and all is well. A bit awkward of an experience, but at least they saved a tree.

The next day, the office worker goes out and finds a different food truck and places an order. They insert a credit card into the reader, and are anticipating having to enter their email address again. But this time, they get a different experience. After inserting the card, the machine says, "Would you like me to send your receipt to the email address we have on file for you?"

What's that? The office worker has never been to this food truck before. But, somehow, the food truck knows that they like their receipts sent to their email address, and knows what their email address is!

Is this a new type of creepy stalking? No, it's a lot simpler than that. Both food trucks are using the same payment network—probably either Stripe or Square. Even though the office worker had never used a credit card at *this* particular food truck, they've used it at a retailer that uses this payment network before. And suddenly, the world is a little bit easier for them because their credit card seemingly knows a bit more about who they are and what their personal preferences are than it did the day before. The office worker gets a customized experience tailored to their preferences.

This is what happens when *network effects* impact credit card processing. Major payment processors, such as Stripe and Square, take advantage of having a certain number of customers (food carts, restaurants, and retail stores), and knowing that their customers' customers (the office worker) all like to have their own personalized experiences. The office worker doesn't have to have eaten at Joe's Diner for the diner to know that when they use a given credit card, they want their receipt sent to their home email address, and when they use a different credit card, they want the receipt sent to their work address for their expense report.

All that's required for this better world to be ubiquitous is for everyone to use the same credit card network. Or, better yet, use a set of compatible credit card networks that all work together toward a common goal of providing a similar personalized experience. This better world is in the process of coming to the payment networks.

Wouldn't it be great if something similar happened with identity? What if, when you logged into your favorite website, that website instantly knew some of the things you liked and could use those preferences to give you a customized experience?

Now, I'm not talking about the creepy/stalking knowledge that happens across advertising networks and social media. I'm talking about a real, shared, personalization network that takes *your preferences*, which *you specify*, and is able to share them across a broad network of websites that *you have chosen to utilize*.

As modern applications start thinking about identity and the network effects of cross-application identity, similar to what credit card processing networks are doing, we can imagine a world where managing identities becomes simpler, and sharing information that you want to share between applications becomes easier and more automatic. We can imagine all sorts of first-world problems disappearing and being replaced with a better overall user experience.

This is the true personalized internet.

Summary

Identity is a critical component of all modern applications, and everyone who interacts with a computer has to deal with identity. Identity is composed of authentication (identifying who a user is), authorization (granting permission to users), and profiles (application information unique to the user). Larger and more complex applications require trust systems to scale the ability to determine authority to perform specific tasks.

As time goes on, advanced identity management for applications is getting more critical as security, privacy, scalability, and usability become increasingly important. For individual users, identity management using tools such as password managers is important in keeping our accounts safe and secure. Modern identity management is a core component of all our lives and our applications.

Our complex applications are requiring a greater focus on identity. As we continue to evolve our applications to provide greater value to our customers, we must also evolve our identity systems to provide greater security and ease of use to our customers. Modern identity management technologies continue to evolve to improve both our applications and our customers' experiences.

About the Author

Lee Atchison is a recognized industry thought leader in cloud computing and the author of the best-selling book *Architecting for Scale* (O'Reilly), currently in its second edition. Lee has 34 years of industry experience, including eight years at New Relic and seven years at Amazon.com and AWS, where he led the creation of the company's first software download store, created AWS Elastic Beanstalk, and managed the migration of Amazon's retail platform to a new service-based architecture. Lee has consulted with leading organizations on how to modernize their application architectures and transform their organizations at scale. Lee is an industry expert and is widely quoted in publications such as *InfoWorld*, *Diginomica*, *IT Brief*, *Programmable Web*, *CIO Review*, and *DZone*. He has been a featured speaker at events across the globe from London to Sydney and Tokyo to Paris, and all over North America. You can find Lee on LinkedIn at *https://www.linkedin.com/in/leeatchison*.