Access Control Use Cases

Alan H. Karp (alanhkarp@gmail.com)

You are very excited. You just got invited to a state dinner at the White House. On the day of the event, you don your finest and head to the venue. A person at the door asks your name and checks a list. If your name is on the list, you're allowed to enter. If not, you're sent home. People have been using this kind of access control list for centuries.

You are very excited. You just got a ticket to the hottest concert of the year. On the day of the event, you don your finest and head to the venue. A person at the door asks for your ticket. If you have one, you're allowed to enter. If not, you're sent home. People have been using this kind of access token for centuries.

Did you notice the difference? To get into the state dinner, you presented something about yourself. To get into the concert, you presented something about the event. That may not seem like an important distinction, but it makes all the difference.

This very simple use case of controlling access to a venue nicely illustrates the dilemma facing computer scientists back in the days when people were just starting to share computers. Before that time, you brought your programs and data with you, had sole access to the computer for a while, and brought your programs and data with you when you were done. Once people started sharing the computer, access control became important. After all, you wouldn't want me using your programs and data without your permission. The question was whether to use access control lists (ACLs) or access tokens.

The most common use case back then was you running a program you wrote with data you provided. There was only one interested party, you. That being the case, there didn't seem to be much difference between ACLs and access tokens. Since the military was the driving force behind computer security in those days, and the military used physical access lists all the time, computer scientists decided to use them. That decision stands today. Any computer you have used almost certainly does access control with ACLs. Windows, MacOS, Linux, even MVS for you enterprise folks, all use ACLs. As you'll see from the use cases presented here, we're still paying the price for that decision.

In what follows, we'll look at a number of use cases and compare how they work with ACLs and access tokens. Each use case ends with a connection to actual events.

An Aside on Nomenclature

There is some controversy among computer scientists over what to call access tokens. When they were first introduced in the 1960s, they were called *capabilities*, because possessing one gave you the capability to do something, *e.g.*, get into the concert. However, the word capability is more commonly used to mean other things, such as “This vehicle has the capability to go 0-60 in 3 seconds.” Some people felt that a new term was needed to avoid confusion.

The issue came to a head around the peak of the popularity of object oriented programming. Since a reference to an object in a programming language, such as Java, serves the same purpose as a capability, people started using the term *object capability*, or *ocap* for short.

However, the term ocap can be misleading, since there are many representations of access tokens that are not object references. Anything that designates a specific permission on a specific thing can be used as an access token. For example, the URL of this document (if you’re reading it online) is an access token granting you permission to read it.

So, what term should we use? I’m going to stick with access token. Who knows? It might even catch on.

Use Case 1: Read a Document

One of the things you use computers for is to read documents people send you. Click on the icon, the system starts a program which reads the file containing the corresponding document, and a window opens showing you its contents. What could be simpler?

This situation is quite different from the lone programmer in days of yore. You are running a program written by someone else with data provided by yet another person. There are now three interested parties, and their interests don’t always align. For example, the person providing the program might want to snoop on what you are reading, or the person providing the document might want to subvert the program and encrypt all your files. ACLs do a very poor job of access control when there’s more than one interested party. You need to know a bit about what’s happening under the covers of your operating system to understand why.

When you log into your computer, the operating system starts a program that represents you to the computer. This program provides the means by which you tell the computer what you want to do and by which the computer shows you the results. This *user agent* is your embodiment in the computer. The upshot is that your user agent necessarily has all your permissions. In a system using ACLs, that means this program has your identity.

When you click on the icon for the file in a window presented by your user agent, the operating system starts the program that will read the document and assigns it **your identity**. There's really no choice on an ACL system; there simply isn't any other identity to choose. The result is both a little good and a lot bad. The little good is that the program can access any file that has your identity in its ACL. The lot bad is that the program can access any file that has your identity in its ACL. That's why a malicious program, or one that's been subverted by a malicious document, can encrypt all your files.

Things work quite differently on a system that uses access tokens for access control. You still have your user agent that has all your permissions, but things change when you click on the icon for the document. The operating system checks that your user agent has permission to access the file, starts the program that will display it, and hands that program an access token for the document. Now the program runs with only the permissions it needs to do what you asked it to do for you, enforcing what computer scientists call the Principle of Least Privilege. The program can't monitor what you're reading because it doesn't have permission to talk on the network. The document can subvert the program, but the program doesn't have permission to write anything.

Attacks using malformed PDFs have been seen in the wild.

Use Case 2: It's an Emergency

Bob has been working on an important contract for a couple of weeks. It's due in an hour, but he's almost done. He's working late to make sure he doesn't miss the deadline when he gets a call. His wife has gone into labor three weeks early and is on her way to the hospital. Now what?

Fortunately, Carol is still around and is willing to finish the job. Unfortunately, her name isn't on the ACL for the contract. Normally, Bob would contact a department administrator to have Carol's name added, but none of them are around. Bob has only one choice; he gives Carol his userid and password before heading out the door. It's a judgment call on his part. He'll get in trouble for sharing his credentials, but he's likely to get fired for messing up a $50,000,000 contract. Plus, he's taking the risk that Carol might secretly hate him and will misuse all the permissions he just handed her to mess with his stuff.

Bob still has a problem if he were using a system with access tokens, but it is far less significant. He simply creates a token granting access to the contract and hands it to Carol. He still might get in trouble for violating company policy, but the violation is limited to the one contract, not all of Bob's stuff.

Such credential sharing is such a pervasive problem that Ping Identity ran webinars on how to discourage the practice.

A Comparison with the Physical World in Two Stories

In an emergency, Marc asks you to park his car in your garage. You can't do it, so you ask your neighbor to do it for you and tell her to get the garage key from your son.

This first story is so mundane you likely never gave it a second thought.

In an emergency, Marc asks you to copy a file from his computer to yours. You can't do it, so you ask your neighbor to do it for you and tell her to get access to your computer from your son.

This second story is so far from our everyday experience that hearing it often generates chuckles.

Use Case 3: Tax Filing Time

Alice's taxes are complicated, so she hires an accountant who needs information from every place Alice has money or investments. Alice hates having to login to each site to download her tax forms and decides to let her accountant do that for her. Of course, the sites use ACLs, so she has to give her accountant all her login credentials. Everything works well until her accountant's gambling addiction leads him to skim from her accounts.

You may laugh at this example. Who would be so foolish as Alice? Apparently, a lot of people. The site mint.com provides an integrated view of your finances – if you give them all your login credentials. As they say in their FAQ:

> Why does Mint need my login information?
> We need your login user name and passwords so that we can help you organize and manage your accounts. We use this information to establish a secure connection with your financial institution or credit card company. This enables us to download and categorize your transaction information securely and automatically.

What they don't say is that they only need permission to read your account information, but they end up with permission to withdraw your money. If your financial institutions used access tokens, you could give these services only the permissions they need.

It's not just websites. Quicken asks for your login credentials for places from which you want to download your bank statements.

Use Case 4: Updating Your Calendar

You were really pleased to get your promotion and the raise that went with it, but you didn't anticipate how many meetings you'd have to deal with. The sheer number was one thing, but the constant changes are driving you nuts. Fortunately, your new position came with an administrative assistant. You can let him handle your calendar. Since your calendar's access control is based on ACLs, all you have to do is give him your corporate login credentials. Of course, that's a firing offense but so is not getting your job done because you're so busy managing your appointments.

The way things are done on the web these days is far better because it uses access tokens. Say that you get an email invitation to a Zoom meeting that includes a link you click to add it to your Google calendar. When you click the link you're presented with a popup window from Google saying, "Zoom wants permission to update your calendar." and containing buttons to allow or deny the request. If you allow the request, Google creates an access token that Zoom uses to make the update. No need to give Zoom your login credentials.

Managers at Hewlett-Packard nearly universally share their login credentials with their administrative assistants.

Use Case 5: Contractors

You've been fortunate. Your company has grown to the point where it is more cost effective to contract out tasks that are peripheral to your business, such as managing the heating and cooling systems at all your locations. The contracting company's employees need access to some of the data on your corporate computers, so you make an account for each of them and set up the appropriate ACLs. After a while managing the comings and goings of the contractor's employees becomes too much of a burden, so you pay a small fortune to set up a federated identity system that lets the contractor manage its employees. It took several weeks to get things running, but you're relieved not to have to deal with all that anymore. Of course, you need to go through most of that time and expense when you bring on an accounting company to manage your SEC filings.

You can save the time and expense of federating identities if you use access tokens. When you sign the contract for the work, you give the contracting company an access token representing all the permissions needed to do the specified work. That company will create new access tokens for each employee granting the permissions that employee needs. If that employee is a manager, new access tokens can be created for each subordinate.

The example uses managing heating and cooling systems, because that was the source of the Target breach.

Use Case 6: Coordinating a Trip

You finally saved enough for the trip you've been wanting to take for years. You don't want anything as simple as a flight delay to cause problems, such as the rental car company canceling your reservation because you didn't show up on time. You decide it's better to use a travel service to take care of such things for you. You book your flight and rental car and give the service your login credentials at the airline and at the rental car company. They make sure the car rental service knows when your flight is delayed.

When you return from your trip, you find out that all your frequent flier miles and rental car rewards were used by someone you never heard of. A rogue employee at the travel service misused your credentials. It was an ordeal, but the travel service eventually made things right.

Things work much more smoothly with access tokens. You can make the reservations as before, but you can give the travel service access tokens instead of your login credentials. That is likely easier than you may think. If you rent a car online, you will get an email with a URL you click to notify the company of any changes you wish to make. That's an access token you can send to the travel service.

This example comes from the Liberty Alliance, a group that proposed an ACL-based approach to solving such problems.

Use Case 7: A Flat Tire

You are driving down the road when you get a flat tire. You pull over to the side of the road and call AAA. "Someone will be there in 40 minutes." you're told. You'd love to wait at the Starbucks you can see from your car, but you're afraid you'll miss the repair guy. You're in luck. The repair truck shows up 10 minutes early, and you get back on your way, but you still wish you could have gotten that latte.

Imagine if your car used access tokens. You contact AAA and give them an access token to open your trunk where you keep your spare. AAA selects a repair service and sends them the access token. The repair service gives the access token to the person who will change your tire, and AAA sends you a text when your car is ready. All the while you're enjoying your latte.

A valet key that lets someone else drive the car but not unlock the trunk or glove box is a physical example of this kind of limited permission.

Use Case 8: Voice Control

You are really enjoying your new toy, a television remote that lets you control not just your television, but all your IoT devices by voice. It's really cool. The remote uses a third party voice

recognition system that you log into when you set up the remote. Initially you use it to control your streaming service, Hulu, and your new smart thermostat.

There are two ways that the voice service can get permission to use Hulu and control your thermostat. One way is to set up its own account on each of them; the other is to use your credentials.

Let's say the voice service uses its own credentials on behalf of all its customers. It must have all the permissions to do anything any of its customers can do, such as retrieve a viewing history. Say that you ask to see someone else's viewing history. (I'm assuming that's an accident; you'd never pry like that on purpose, would you?) It's likely that the voice service doesn't really understand the requests you make; it just forwards them to the right place. In this case, Hulu uses the ACL for the service, retrieves the viewing history which eventually reaches you. You have just used somebody else's permissions to do something you don't have permission to do yourself.

Since using the voice service's credentials is a bad idea, the only other choice is for it to use yours. Now, the service logs in as you to Hulu and your thermostat. If you ask for someone else's viewing history, Hulu uses the ACL associated with you and rejects the request. Unfortunately, there's still a problem. Say that you ask to see the film "Fahrenheit 451," but the voice service has a bug and sends the request to your thermostat. The thermostat's ACL says you have permission to change the temperature, so you end up a pot roast.

There are no such issues with access tokens. Saying "Hulu: viewing history for John Doe" would select your Hulu access token, which doesn't authorize you to see his history. Saying, "Hulu: watch Fahrenheit 451" could send your Hulu access token to your thermostat, but it won't honor the request.

When the voice service uses its own credentials, it has become a confused deputy, which is a vulnerability that results in many different attacks, such as cross-site request forgery.

Use Case 9: The Cleaning Lady

You are really happy with your new smart door lock. No poking around for your keys; less fumbling with packages getting in the door. You even set it up for the cleaning lady, who would forget to bring your house key with her at least twice a year. Now all she needs to remember is her phone. The ACL on the door lock knows to open for her on Monday mornings.

Everything goes swimmingly until the Monday morning she has to take her daughter to the doctor. What's she to do? She can give her phone to the person she uses as a substitute, but then she won't have her phone that she relies on for *everything*. The other choice is to skip your house and have a disappointed customer.

Wouldn't it be nice if the door lock accepted an access token? You cleaning lady could simply generate a new token, give it to her substitute, and be done with the matter.

There are smart door locks that include a keypad. You can program a new code when you need to let someone into your house and delete it when the work is done.

Use Case 10: Copy a File

It's been a long day, but you finally finished the edits on your manuscript. All that's left is to copy your working version to the publication system. That application gives you a menu where you say what files you're talking about. You fill in the fields on the menu, click OK on the warning that you already have a file by that name. — And then you groan. You entered the file names in the wrong fields. You just replaced the one with all your new work with the old version. Why would the system let you do something that stupid? Well, how would it know? It's not a mind reader. The ACL says you have permission to read and write both files, and you have no way to tell the system that you only want to read the version you've been working on. The good news is you have a backup and only lost an hour's work.

With access tokens, it's possible to provide an interface that lets you select the exact permission you want to use. If you specify an access token with only read permission in a place that needs a write permission, your request will fail, exactly what you want to happen. Can you still get it wrong? Of course, but at least access tokens give you a fighting chance.

The author has made this mistake – more than once.

Use Case 11: A Publication Service

Your company just signed up to use a publication service that provides some really nifty features. You like everything about it except the billing system. It's not that you mind paying out of your department's budget (Well, you do, but that's a different matter.) It's that you always run out of budget before the end of the month. When that happens, you can't do anymore updates. You can't help thinking how much easier your job would be if you could just replace the billing record with one showing you still had funds.

After a little research, you figure out how you can. Submitting a request involves specifying a file on your machine and a file on the publication system to receive it. Each time that happens, the publication service writes a little file with your new account balance. The ACL doesn't give you permission to write that billing file, but it does give that permission to the service. So, the next time your budget runs low, you submit a request specifying a file containing a bogus balance as the source and the billing file as the target. The service happily reads what you provide and writes as you directed. Since the service has permission to write the billing file, you just bypassed your budget limitation. Management is Very Angry™ when they find out, but they agree to increase your publications budget anyway.

That trick won't work on a system using access tokens. Even if you had an access token to the billing file, it wouldn't give you permission to write it. If you try to fool the system, you provide a read access token for the source file and your read access token to the billing file. The service will use these tokens, which means the attempt to write to the billing file will fail.

An example similar to this one occurred in an early time sharing system, sort of a 1980s version of cloud computing. Norm Hardy identified the source of the problem and coined the term confused deputy.

Use Case 12: The Professor

You might actually be able to get tenure after all now that you got that PlanetLab grant. You and your students can do really large scale computations by combining thousands of computers from dozens of data centers into a single computational cluster. All you have to do is set up an account in each data center for each of your students. No big deal you think until the start of a new semester, and you have to make a ton of changes at each data center for the students that left and the new ones now working for you. Oh yeah, and if you forget to remove an account for a student who left, you're personally responsible for any charges they run up, on an assistant professor's salary no less.

Things would be so much simpler with access tokens. Each data center gives you an access token when you get your grant. You create a new access token for each student that expires at the end of the semester. It doesn't get much simpler than that.

As told to the author at a PlanetLab workshop.

Use Case 13: Nobody Knows How to Make a Pencil

In 1958 the economist Leonard Read published an essay, "I, Pencil." In it he uses the example of a simple pencil to describe how specialization leads to economic efficiencies. He points out that one company deals with the wood, another the graphite; two more provide the eraser and the gizmo that holds it on. Your company assembles the parts.[1] Imagine how inefficient it would be if pencil users had to do all those things themselves.

Unfortunately, all the major cloud service providers that use ACLs have to build their own "pencils." Let me explain with Netflix as an example. When you click on something in Netflix, anywhere from a dozen to a couple of hundred small applications, called microservices, get invoked. Many of these microservices need to access data on your behalf, so the ACL requires that they each be able to use your identity. That's fine if Netflix runs all the services, but it makes it hard for Netflix to outsource any of them. You'd be really angry at Netflix if one of

[1] Or, as the joke goes: "What's in the box?" "Some wood, a piece of lead, an eraser, and an allen wrench." "Oh, somebody bought a pencil at Ikea."

those business partners misused your permissions. It's simpler for Netflix just to do everything itself. In effect, it's making its own pencils.

There wouldn't be nearly as large a barrier to outsourcing if Netflix used access tokens. Netflix could give each outsourced microservice an access token for the exact set of permissions needed to do the job it is assigned. There's still a risk that a bug or compromise could result in misuse of this subset of your permissions, but the harm is far less than if the microservice was able to impersonate you.

It's not just Netfix, of course. Every large cloud service provider is in the same boat. The problem is large enough that it probably has a noticeable negative effect on each company's bottom line.