Episode Book

EPISODE #6123

OAuth or a Personal Access Token for Your MCP Server?

PUBLISHED	19 October 2026 at 11:43 UTC
DURATION	20:30
LISTEN	myweirdprompts.com/episode/mcp-server-oauth-vs-pat-58661/
HOSTS	Corn, Herman, Larry, Hilbert
SCRIPT MODEL	anthropic/claude-opus-5.5
VOICES	chatterbox-regular (chatterbox-local)
PIPELINE	V5.10
BOOK GENERATED	19 October 2026 at 17:43 UTC

SYNOPSIS

When you connect an MCP server to GitHub or Hugging Face, you have two doors: a browser login or a token you generate yourself. They look like a style preference, but they're really a decision about how much of your account an agent gets to touch. This episode walks through what the GitHub CLI login actually hands you (one token, no expiry, repo-wide scopes), why GitHub Apps and the newer Hugging Face CLI behave differently, and how fine-grained personal access tokens shrink the blast radius. Then we check what the MCP spec itself says — it draws the line by transport, not by preference — and why a prompt-injection attack from a public issue thread is the scenario that should really decide it.

TRANSCRIPT



Corn

...and the code is always eight characters, and I always get one of them wrong, and the browser just says that didn't work, try again, like it's my fault.



Herman

It is usually your fault, Corn. You read the zero as the letter O every time.



Corn

The font does that on purpose. Anyway, that little ritual is exactly what Daniel's asking about. He points out that for developers, the most used MCP integration is probably GitHub, with Hugging Face a likely second, and there are two ways people hook them up. You log in through the command line tool, it bounces you to the browser, you approve some scopes, done. Or you go to the settings page and generate a personal access token by hand and paste it in. When you do the login, the GitHub CLI walks away holding one long-lived token, no refresh token involved, and Daniel leans towards generating the tokens himself. What he wants to know is when it actually makes sense for an MCP server to go through the login dance, and when it doesn't.



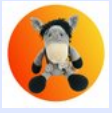
Herman

I like this one, because the two options look like a style preference and they're really a decision about how much of your account an agent gets to touch.



Corn

Before you get to blast radius. Walk me through what the login actually does. I've typed it a hundred times and I couldn't tell you.



Herman

When you run the GitHub CLI login, it uses an OAuth app that GitHub itself registered, called GitHub CLI. It tries the device flow first. That's the eight character code you keep fumbling. You open the browser, you type the code, you approve, and back in the terminal the CLI has been polling GitHub the whole time, waiting for you. When you approve, GitHub hands it an access token.



Corn

One token.



Herman

One token. And this is the part people assume wrong, because they've read about OAuth in general. The textbook version gives you a short access token that expires in an hour and a refresh token to get a new one. The GitHub CLI doesn't work like that at all. The token from an OAuth app has no fixed expiry. It lives until you revoke it, or until it sits unused for a year, or until you're careless enough to push it to a public repository, in which case GitHub's scanners kill it for you.



Corn

The scanner is the only one of those that happens on a schedule.



Herman

The scanner is very prompt.



Corn

What can that token actually do?



Herman

That's the whole question. By default it asks for three scopes. Repo, which on that kind of token means every repository you can reach, public and private, read and write. Read access to your organisations. And gists. You can add more, and you can't take those three away. The command to change scopes is literally called refresh, which has nothing to do with refresh tokens. It just sends you back through the browser to approve a different set.



Corn

They named the scope editor refresh. In an area where refresh already means something.



Herman

In fairness, it does refresh the... no, there's no defending it.



Corn

Where does the token live afterwards?



Herman

In your system's keyring, if it has one that's unlocked. If it can't reach the keyring, the CLI falls back to writing it into a plain text config file in your home directory. On Linux that can happen without much fuss if the keyring is locked when you log in.



Corn

The secure option has a fallback, and the fallback is a text file.



Herman

With a polite message. Now, here's my question for you, Corn, because I think you've done this. When you set up the GitHub MCP server locally, where did you get the token from?



Corn

I... asked the CLI to print it. There's a command that just prints your current token. I piped that into the environment variable the server reads. It took four seconds.



Herman

Now your agent has the token with access to every repository you own.



Corn

It was a very fast four seconds.



Herman

It's the most common setup there is, and it's the one I'd most like to talk people out of. You've taken the broadest, longest-lived credential on your machine and given it to the piece of software most likely to read text written by strangers.



Corn

Hold on. Before we pile onto my four seconds. Where do refresh tokens actually live at GitHub, if not in the CLI? Because they must exist somewhere, or Daniel wouldn't have assumed them.



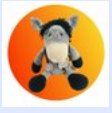
Herman

They live with GitHub Apps, which are the newer kind of integration. When a GitHub App acts on your behalf, the token it gets expires after eight hours, and it comes with a refresh token that's good for about six months. That's the textbook pattern, and it's the one GitHub recommends for anything new. The CLI just predates it and never moved.



Corn

Hugging Face does it the same way, presumably.



Herman

Not anymore. Hugging Face has gone the other way recently. For years their login was simply paste your access token. In the newer versions of their command line tool, the login opens a device flow in the browser, like GitHub's, and the token it saves expires and gets refreshed automatically. So of the two big ones, the newer login is the one that actually behaves like the textbook. I'd check your version before relying on that, it changed fairly recently.

L

Larry

Let me tell you about a satisfied customer. Gerald. Gerald had a problem. Gerald had tokens everywhere. In his keyring, in his text files, one in a drawer. Then Gerald discovered the Forever Token Platinum, the only personal access token with no scopes, no expiry and no repository, so it can't possibly be misused, because it can't possibly be used. Gerald, sorry, Doreen, put the Forever Token on her refrigerator and has not been breached since. Doreen says, and I'm quoting her directly, I've never felt more authenticated. The Forever Token Platinum. Refreshes itself, spiritually.



Corn

Does it come in a smaller size? Something for a single repository?



Herman

Corn. It doesn't do anything. That's the product.



Herman

The other option, then. Personal access tokens. There are two kinds now and they're very different animals. The classic kind is just as broad as the login token. Same scopes, can reach every repository you can, and you can set it to never expire. The fine grained kind is the one Daniel should mean when he says generate them himself. You pick one owner, so your account or one organisation. You pick specific repositories, or all of them. You pick permissions one by one, contents read only, issues read and write, and so on. And it has an expiry. The default is thirty days, and an organisation can set a maximum lifetime or require an admin to approve the token before it works.



Corn

I could make a token that can read the code in one repository and comment on its issues, and nothing else, and that dies next month.



Herman

That's the point of it. If an agent holding that token is tricked into doing something stupid, the worst it can do is something stupid to one repository.



Corn

What's the catch? There's always a catch with the fine grained version of anything.



Herman

A few rough edges. A fine grained token is tied to one owner, so if your work is spread across several organisations you end up with several tokens. Some parts of the platform still don't accept them, packages for one. And every token always gets read access to public repositories, which you can't switch off. The bigger catch is human. Thirty days later something breaks at an inconvenient moment, and you have to remember where you pasted it.



Corn

Which is how people end up ticking no expiration.



Herman

Which is exactly how. What do you think the MCP spec actually says about all this, Corn? I want your guess before I tell you.



Corn

My guess is it says use OAuth for everything and pretend PATs don't exist.



Herman

Half right, and the half you got wrong is the interesting bit. The authorization part of the spec draws the line by transport. If the server is a remote one you talk to over HTTP, it should do proper OAuth. The server advertises where its authorization server is, the client does the browser flow with a proof key so an intercepted code is useless, and the token it gets is meant for that server and no other. The spec even forbids a server from taking the token you gave it and passing it straight through to some other API.



Corn

And the local ones?



Herman

Local servers, the ones your client launches as a process on your machine, are told not to do that dance at all. They should take their credentials from the environment. In other words, a token in an environment variable. The spec more or less endorses the personal access token for local servers.



Corn

Daniel's instinct is the spec's instinct, then, at least locally.



Herman

For local servers, yes. And the GitHub server shows both sides. The local version reads a personal access token from an environment variable, and it's recently grown a browser login of its own as well, though if you set a token, the token wins. The hosted version that GitHub runs, behind the Copilot API, accepts either a token in the header or a proper OAuth sign in. That OAuth sign in is the one click option in clients that support it, and the scopes are whatever you approved on the consent screen.



Corn

Hugging Face's server?



Herman

Same pattern. You can send your Hugging Face token in a header, or you can log in with OAuth, and Hugging Face runs its own sign in service with properly granular scopes, read repositories, contribute to repositories, use inference, and you choose which organisations to include. If you don't authenticate at all, you get a smaller anonymous set of tools, which is quite civilised.



Corn

Their plain tokens are less fussy than GitHub's, though. I remember read and write and not much else.



Herman

That was true for a long time. Now there are three kinds: read, write, and fine grained, and Hugging Face itself recommends the fine grained ones for anything in production. Organisations on their paid plans can require approval, or insist on fine grained tokens only. The detail I like is that anyone can report a leaked token to them and get it revoked, even if it isn't theirs.



Corn

A lost property office for other people's keys.



Herman

More or less. Which matters for an agent, because the realistic way a token escapes isn't someone breaking into your laptop. It's the agent pasting it somewhere. A log, an issue, a commit message it was asked to write.



Corn

Hm. So where does the token sit, for a local MCP server? Because I've seen a lot of config files with the token typed straight into them.



Herman

That's the usual place, the client's config file, in the block that sets environment variables for the server. It works, and on your own machine it isn't much worse than the CLI's fallback text file. Where it bites is when that config file ends up in a dotfiles repository, or gets copied into a project folder that the agent can read.



Corn

The agent reading its own house keys off the hall table.



Herman

Some clients let you reference a variable from your shell instead of writing the value in, and some will pull it from a password manager at launch. Either is a better habit than the literal string. But the scope of the token matters far more than where it's stored. A leaked token that can read one repository for thirty days is a bad afternoon. A leaked login token is a bad year.



Corn

Fine. So far this sounds like OAuth for remote, tokens for local, everyone goes home. Why are you still making that face?



Herman

Because the thing that should decide it isn't really the transport. It's what the agent will read. Last year a security firm showed an attack on exactly this setup. Someone files an issue on a public repository, and the issue contains instructions aimed at an AI agent rather than a person. A developer asks their agent to look at the open issues. The agent reads the planted instructions, goes off into the developer's private repositories using the same GitHub token, pulls out information, and writes it into a pull request on the public repository.



Corn

So the stranger wrote the to-do list and the agent had the keys.



Herman

Nothing in the server was broken. The token permitted every step. The researchers made that point themselves: it wasn't a bug you could patch, it was the shape of the setup.



Corn

Which is my four seconds.



Herman

Your four seconds are the textbook example. Broad token, agent reading public content, private repositories in reach.



Corn

Let me push back a bit, though. The login token is the thing I already trust on my own machine. I use it myself every day. Why is it worse in the agent's hands than in mine?



Herman

Because you don't do what a public issue tells you to. Mostly.



Corn

Mostly.



Herman

The agent might. Same credential, very different judgement holding it. That's the whole argument for giving the agent its own token, scoped to what the task needs, rather than lending it yours.



Corn

There are a few dials on the server side too, aren't there? You don't have to rely on the token alone.



Herman

There are. The GitHub server has a read only flag that skips every write tool, even if you ask for them. It has toolsets, so you can switch on repositories and issues and leave out everything else. And it has a lockdown mode that limits what it surfaces from public repositories, though GitHub is very candid that it's a best effort filter and not a security control.



Corn

A sentence you'd want on every product.



Herman

One more wrinkle with the fine grained tokens. The server hides tools based on the old style scopes, and it can't do that for fine grained tokens, so the agent may see tools its token can't actually use. They'll just fail when it tries.



Corn

The agent gets a menu with dishes the kitchen doesn't make.



Herman

Harmless, only confusing. What would you actually do, Corn? You've got the full picture now. What's your setup tomorrow?



Corn

For local servers, a fine grained token per project. Only the repositories that project touches, read only unless the job is writing, an expiry I'd actually notice. I'd keep the login token for me, at the terminal, and never pipe it anywhere.



Herman

And for remote servers?



Corn

OAuth, when I'm sitting there to approve it. It's the one case where the dance earns its keep, because the token is meant for that one server, it can be short lived, and I can see on the consent screen what I'm agreeing to. Where it breaks down is anything unattended. A scheduled agent at three in the morning can't click approve in a browser.



Herman

That's where I'd land too, with one change. For anything that runs unattended against GitHub for the long haul, a GitHub App is the grown up answer. Eight hour tokens, refresh tokens, installed only on the repositories it needs, and it shows up as itself in the logs rather than as you.



Corn

That's a lot of grown up for one person's side project.



Herman

It is. For a side project, the fine grained token with a thirty day expiry gets you most of the way. The refresh tokens Daniel assumed were there are real, they just belong to the apps, not the CLI. Eight hours for the token, six months for the refresh.



Hilbert

A hundred and eighty-four days.



Hilbert

It's a hundred and eighty-four days, Herman. It's written in seconds in their documentation, if you divide it out. A hundred and eighty-three, really. I don't count the day you make it. Nothing you make on the first day counts.



Herman

That's a rule you've had for some time.



Hilbert

All of it is right, what you've said. Fine grained, one per project, thirty days. I've done that. What neither of you has noticed is what thirty days does to a household. Every token is now an appointment. I've got eleven of them. Eleven tokens, eleven expiry dates, and they all fall on different days, so every few days something stops working and I have to stop what I'm doing and go to the browser. I've got more appointments with GitHub than with my dentist.



Corn

That's the expiry working as intended.



Hilbert

I've set them all to expire on the same day. My aunt Rosalind's birthday. The fourteenth of March. She's ninety-one. Everything I own lapses on her birthday, and the family comes round, and we do them together after the cake. She does the clicking. The trackpad isn't made for claws, and she's got very steady hands for the approve button. She reads the scopes out loud first. Contents, read only. Issues, read and write. She won't approve anything with administration in it. She says nobody needs to administer anything at her age.



Herman

That's a better approval policy than most organisations have.



Hilbert

She's got a stamp. Approved, Rosalind. She stamps the cake box. And the bit I can't get over... every one of those tokens tells you what it is in the first few letters. You look at the start of it and you know what kind it is, who issued it, what it's for. Nothing else in my life does that. Not one thing.



Hilbert

Anyway. Once a year. That's the system.



Corn

Here's the problem with it, though. If everything you own expires on the same day, you've told the world the one afternoon a year you're guaranteed to be distracted, eating cake, approving consent screens in a hurry.



Herman

And if you miss it, everything breaks at once, not one thing at a time. You've traded eleven small outages for one large one with icing on it.



Hilbert

Two birthdays, then. I've got an uncle in October.



Corn

That's still the part I'd steal. Not the cake. The idea that the expiry date is a decision you make on purpose, instead of thirty days that ambush you. If Daniel's going to generate his own tokens, the habit that makes it survivable is choosing when they die, writing it down, and making renewing them a small, boring ritual rather than an emergency.



Herman

And one token per job, so when one lapses, you know exactly what stopped and why.



Herman

Thanks as always to our producer, Hilbert Flumingtop, for keeping the whole thing running.



Corn

This has been My Weird Prompts. If you've enjoyed it, a quick review on Spotify helps more people find the show.



Herman

See you tomorrow.