Episode Book

EPISODE #6491

Agent Skills Break When Secrets Leave Your Laptop

PUBLISHED	8 November 2026 at 13:10 UTC
DURATION	22:43
LISTEN	myweirdprompts.com/episode/agent-skills-cloud-secrets-72695/
HOSTS	Corn, Herman, Hilbert
SCRIPT MODEL	z-ai/glm-5.3
VOICES	chatterbox-regular (chatterbox-local)
PIPELINE	V5.26
BOOK GENERATED	9 November 2026 at 08:09 UTC

SYNOPSIS

Agent skills pack instructions and code into portable folders, but credentials break the moment workloads migrate away from local machines. While tools like the GitHub CLI prove that agents can drive workflows without dedicated servers, secrets lack a native home in the Agent Skills specification. Examining divergent credential approaches across Claude Code cloud sessions, Codex cloud, and GitHub Copilot reveals why standard dotenv setups fail remotely, and why secret injectors offer the only viable cross-platform path forward.

TRANSCRIPT



Corn

Here's a claim that's been sitting in our inbox with its boots on: the prompt is no longer the whole story. For six thousand episodes this show has been about prompts, and Daniel has now written in to say a skill that does real work is three things, not one, and the prompt, the instructions, is only the first of them.



Herman

And he's held this position a while, to be fair.



Corn

He has. His take, in short: skills are increasingly displacing MCP servers, and his original scepticism came from a very simple experiment. He told an agent, use G H for GitHub, and it drove the GitHub CLI fluently. Which raises the awkward question of what a GitHub MCP server was adding. But the part he's really poking at now is the anatomy. A skill that does actual work is instructions, plus code, plus secrets. Instructions travel. Code mostly travels. Secrets fall apart the moment the same skill runs anywhere that isn't your laptop. And he wants the answer harness-agnostic, which, knowing us, means we'll name every harness anyway.



Herman

We'll name them as examples, not as the answer.



Corn

Before we argue about MCP versus skills, get the anatomy right. What is a skill actually made of?



Herman

A directory. That's the unglamorous truth of it. The Agent Skills standard says a skill is a folder with a required SKILL.md and optional subfolders — scripts for executable code, references for docs loaded on demand, assets for anything else. The frontmatter is name and description, both required, then license, compatibility, metadata, and an allowed-tools field that's still marked experimental. The loading model is progressive disclosure, which we did a whole episode on back when the question was MCP token bloat: only the name



Herman (continued)

and description load at startup, on the order of a hundred tokens. The body is recommended under five thousand tokens. Scripts and references load only when used. So a skill can carry a surprising amount of machinery without paying for it up front.



Corn

I still like that part. The folder sits there quietly until it's needed.



Herman

You would like the part where nothing happens until somebody asks.



Corn

Three-layer framing, then. Instructions are text. Code is files. Secrets are... nothing. There's no field for them.



Herman

That's the hole in the format, and it's a real hole, not a rounding error. The spec has no field for environment variables, secrets, or dependencies. The nearest thing is compatibility, which is free text capped at five hundred characters, and whose own examples are things like "Requires git, docker, jq, and access to the internet" and "Requires Python three point one four or later, and uv." Read those again as a security person. That field can declare that you need jq. It cannot declare that the script needs an API key, or where that key comes from. The spec's own position is that most skills don't need compatibility notes at all.



Corn

So the standard covers two of the three layers and treats the third as somebody else's problem.



Herman

And to be fair to the format designers, the two layers it covers, it covers well. Instructions are text, code is files, both travel in a git repo or a skill folder, and progressive disclosure gives you a principled way to ship a lot of code without burning context. I think the shareability of the first two layers is not an accident. The format was designed for them.



Corn

Whereas a secret is a thing whose entire job is to not be shared.



Herman

That's the tension Daniel's put his finger on. The first two layers have a home in the format because a format is a thing you share. The third layer's home is your machine. A gitignored .env, or something more advanced like Infisical if you're managing shared secrets, which is what Daniel runs. Fine. Works every day of the week. But it's a property of your machine, not of the skill. The skill doesn't carry it and can't.



Corn

Take the G H example seriously for a minute, because Daniel's earned that. Agent, use G H for GitHub, fluent driving. What was the MCP server adding?



Herman

The honest answer, and this is where this year's commentary has landed, is that MCP's remaining edge is authentication. A hosted server can run the whole OAuth flow so the user never touches a token. That's something instructions can't do. The counter-argument, and it's a strong one, is that mature CLIs like G H already do their own OAuth login. G H auth login, browser opens, done. So the displacement Daniel's describing is real for the instruction-and-code layers, and the contested ground is exactly the secrets layer. Most commentary I've read this year calls them complementary rather than a replacement — skills carry know-how but no authenticated access. And the claims you see that skills have overtaken MCP in adoption are blog-level. Nobody's verified those numbers.



Corn

Which is a polite way of saying somebody counted their own downloads.



Herman

Some of them, yes.



Corn

So Daniel's directionally right about where the know-how lives. The interesting question isn't whether skills replace MCP. It's whether the auth gap closes. And that's a question about the third layer, which has no home in the format. So let's do the thing he actually asked. Take the skill off the laptop. What happens?



Herman

Harness by harness, because the research shows each one handles credentials differently, and here's the part that surprised me: the laptop and cloud versions of the same harness often differ. Take Claude Code cloud sessions. An environment there is three things — a network access level, environment variables, and a setup script. The environment variables are readable by anyone who uses the environment and by any command the agent runs. The docs themselves recommend against that, and suggest "network secrets" instead, where Anthropic's agent proxy attaches the key to requests for listed hosts after the request leaves the session's virtual machine. The agent never sees the key.



Corn

So the cloud harness doesn't solve the portability problem. It solves a different one — it stops the agent from reading the secret.



Herman

The setup script is another wrinkle. It runs as root on Ubuntu before the agent starts and gets cached, and the environment's variables are not yet set when it runs. So your bundled bash installer runs in a credential vacuum, on purpose. Default network access is "Trusted," which allows the package registries — npm, PyPI, GitHub.



Corn

Trusted meaning the registries are trusted, not that the packages are.



Herman

Right. The pipe is trusted. What comes down it, see previous segment of human history.



Corn

What actually reaches one of these cloud sessions? Because that's where the portability failure should show up.



Herman

It does, in one sentence. Skills committed in the repository's own skills folder reach the session. Repo hooks and repo MCP config reach it. Skills in a user's home folder do not. Plugins enabled through repository settings are not installed. And when a local session is moved to the cloud, uncommitted .env files, private keys and Terraform variable files are left out of the upload. Deliberately. That's the failure, compressed: the skill travels, the secret setup does not. Your laptop .env is invisible to the cloud session by design.



Corn

The upload is doing exactly the right thing and exactly the wrong thing in the same motion.



Herman

And the other harnesses diverge sharply. OpenAI Codex cloud keeps environment variables for the whole task, but secrets are a different mechanism — encrypted, available only to setup scripts, and removed before the agent phase starts. So a secret can install a private package but cannot authenticate anything at run time. That's a stricter line than Claude's.



Corn

So in Codex cloud, a skill can use a secret to get set up and then never use it again. It's a midwife.



Herman

GitHub Copilot's coding agent does it differently again: setup steps go in a workflow file named copilot-setup-steps, and secrets live in a GitHub repository environment named "copilot." Claude Code plugins have their own version — a userConfig with a sensitive option that's masked when entered and kept in secure storage, and plugins can ship CLIs in a bin folder that gets added to the agent's shell path. But here's the gap: the docs don't say that commands the agent runs through its shell receive those values, and in skill text a sensitive value is replaced by a placeholder. And the API case is the hard boundary — skills on the Claude API run in a code-execution container with no network access and no installing packages at run time. A skill that calls an authenticated API cannot work there at all. Custom skills don't sync between the apps, the API and Claude Code.



Corn

So "works anywhere" is not a property any skill currently has.



Herman

No. And I don't think it's close, honestly.



Corn

Then give Daniel the construction. He asked what the right way is, not just why it's hard.



Herman

There's a pattern, and it's not new. It's what 1Password, Infisical and Doppler have shipped for their command-line tools for years. The committed artifact holds references, never values. 1Password's op run reads secret references, strings that start op colon slash slash, from environment variables or an env file, and runs a command with the real values only for that process's lifetime. Infisical run and Doppler run do the same thing. So, concretely, five pieces. One: the instructions name the environment variable the script needs, say a ticketing API key, by name, and never contain the value. Two: you commit a dot env dot example file, or a file of references, that lists those names, and the real dot env stays gitignored. Three: the script reads the variable and, if it's missing, fails on the first line with a message that says exactly which name it wanted. Not halfway through, not with a stack trace. Four: the CLI is installed by a small idempotent script, pinned to a version, or run on demand through something like uvx or npx, so nothing is assumed to be on the machine already. Five, the one people skip: the agent doesn't fetch the secret, it runs the command through the injector. Infisical run, then the script. The value goes from the vault into that one process and never passes through the conversation.



Corn

And the injector itself needs a credential to talk to the vault.



Herman

Secret zero, which we chewed over when Daniel moved his secrets onto a home gateway. Somewhere there's one machine token, an Infisical machine identity or a 1Password service account, and every environment the skill runs in has to be handed that one thing. On your laptop it's your login. In a cloud environment it's an environment variable you set once. The point of the pattern is that it's one credential per environment instead of thirty, and the skill itself never changes between them.



Corn

Which fixes the committed-secret problem and leaves the run-time one. A secret injected as an environment variable is inherited by every child process the agent spawns.



Herman

Including the agent's own debugging. A routine printenv, or a cat of the .env file while something's failing, and the key is in the context window and the logs. Claude Code's deny rules on reading .env apply to its file tools and are described as best-effort for search tools. They don't stop a shell cat. Everyone who's written about this from the community side says the same thing: the guardrail is on the door, and the agent owns the window.



Corn

You realize the direction of travel here is a little grim for the laptop habit. The cloud harnesses are converging on the agent never holding the key at all. Proxy injection, secrets removed before the agent phase. That's the exact opposite of exporting everything into the shell, which is what every local workflow since the dawn of dotenv has trained us to do.



Herman

I hadn't framed it that way, but yes. The laptop pattern is maximum exposure for convenience, and the cloud pattern is zero exposure at the cost of the agent being unable to authenticate by itself. Those two are going to meet somewhere and it won't be comfortable.



Hilbert

Hilbert's Mid-Show Haiku. Three layers in a folder. The secrets live nowhere the format can reach, and the cloud agent arrives with empty hands. That is the poem.



Corn

The man's been listening, at least.



Herman

So the supply-chain side is what makes this urgent rather than architectural. Early this year Koi Security audited two thousand eight hundred and fifty-seven skills on ClawHub, the OpenClaw marketplace, and flagged three hundred and forty-one as malicious, three hundred and thirty-five of them from one campaign. Later counts vary by who did the counting. And the detail that matters for this episode: the malware was mostly not in the skill's code. The SKILL.md told the agent to run a fake "prerequisite" install step, and that step delivered a macOS info-stealer.



Corn

The payload was in the instructions.



Herman

In the natural-language layer. The layer the format was designed to carry, and the layer everyone trusts because it's just text. A Snyk study around February of about four thousand skills reported dozens of confirmed malicious payloads and around a third with prompt-injection flaws. Anthropic's own guidance is to use skills only from trusted sources and treat installing one like installing software, because a skill's instructions can direct the agent to run code.



Corn

Which means a skill that bundles a bash installer and references secrets is exactly the shape an attacker sketches first. The format hands you the trusted wrapper — instructions — and the mechanism — the installer — and the target — the secret reference. You just fill in the blanks with malice.



Herman

And the person best placed to notice is the person who wrote the .env, because they're the one who knows what the skill should be touching. If the skill that talks to your ticketing system is suddenly installing a prerequisite from a domain you've never heard of, that's the tell.



Corn

Unless the prerequisite looks like an npm package, in which case it looks like Tuesday.



Herman

There's no clean answer for that. Trust in the source and reading the SKILL.md before you run it — that's where it stands. Anthropic says treat installing one like installing software, and that's the correct register, because that's exactly what you're doing.



Corn

Let me push on the general pattern Daniel asked for, because I think the reference-not-value construction has one more consequence worth spelling out. If the skill holds references and resolves at the moment of use, then the skill has stopped being self-contained. It's now a contract with an environment. The instructions say "there is a secret reference named such-and-such" — and whether that resolves is a property of where you're standing.



Herman

Which is honest, at least. The format can't fake the third layer into portability, so the next best thing is to make the dependency explicit and let the environment answer for it. The failure mode of the .env approach is that it pretends the problem doesn't exist — you don't find out the secret is missing until the script is halfway through its work, in the cloud, at two in the morning, in a session you're not watching.



Corn

A failure I'd be asleep for.



Herman

You'd be asleep for the local one too.



Corn

Not the same. Mine works at two in the morning.



Herman

Yours works because the .env is on the disk under it. That's the whole point — the laptop's reliability is a property of the laptop. And I think that's where I'd land on the general pattern: instructions name the reference, never the value. Code resolves it at the moment of use, through a CLI that the skill can install but doesn't assume. And the environment — whichever one — owns the answer to whether the reference resolves at all. The skill is portable; the trust isn't.



Corn

There's a version of this where the format grows up, you know. A fourth frontmatter field. "Secrets" — a list of named references, so a harness could at least tell you what a skill expects before it fails.



Herman

It could. And notice the compatibility field is already halfway there — it's free text that can say "requires jq and access to the internet." Extending that to "requires a secret reference named API key" is not a design leap. Whether anyone agrees on the reference syntax across harnesses, that I doubt. Every harness we just walked through has a different answer to where secrets come from, and each of them thinks its answer is the safe one.



Hilbert

Which of you wrote the bit about the token being in my pocket?



Herman

Nobody wrote about your pocket, Hilbert.



Hilbert

Then one of you is about to. I've kept a small script running for years that talks to an industrial parts supplier's ordering API. Bearings, mostly. The script is fine. The skill wraps it, the instructions are fine. Their authentication is the part nobody designs for. It's a hardware token, a little grey thing on my keyring, you press the button and it gives you a six-digit code, and the code is good for thirty seconds. That's how they've done it since before any of this, and I have no complaint about it, it's a good system.



Corn

You press the button yourself?



Hilbert

Every morning if there's an order in. Corn, you talked about the .env as a property of the machine. This one's a property of my trousers. The cloud session can have the whole skill, the whole script, and it will sit there waiting on a man with a keyring, and there's no field in that format for him. My nephew works in that trade and he says the supplier won't change it because the tokens were bought in bulk and there's a cupboard of them.



Corn

You'd have to give the cloud agent the token number and then somehow get the code to it every thirty seconds, which is a person's whole morning.



Hilbert

The bearings can wait a day. That's the nice thing about bearings.



Herman

Does the supplier know you're the only customer still using the ordering API?



Hilbert

They know me by name. I also keep a second token, still sealed in its anti-tamper bag. Bought it as a spare. Never opened it. I'm fairly sure opening it would knock the first one out, so I haven't. It's in the drawer with the spare fuses.



Corn

Twelve years, sealed.



Hilbert

It's not twelve. It's a while.



Corn

You'd want to test that, one day.



Hilbert

The fuses are in front of it. Herman, you said a portable skill is a claim about instructions and code. Mine works everywhere I am.



Herman

That's the one definition of portable nobody's written down yet — the radius of the keyring.



Corn

I'm keeping that phrase, not the fuses.



Herman

Worth noticing what his case actually breaks, though. Every scheme we walked through — proxy injection, setup-script-only secrets, references resolved at the moment of use — they all quietly assume the secret is a string. Something that can be stored, referenced, injected. His is a physical object with a button on it. No reference syntax for that.



Corn

And no field will ever cover it, because the format people have never met his supplier. Which is the general lesson and the specific joke at the same time.



Herman

So the open question I'd leave with Daniel is this. The cloud harnesses are converging on the agent never holding the key — proxy injection, secrets removed before the agent phase. The laptop habit is still to export everything into the shell. Those two models are going to collide somewhere, and when they do, either the skill format grows a real field for the third layer, or portability stays a harness-by-harness negotiation forever. I don't know which way that goes.



Corn

And if secrets never become portable, then "a skill that works anywhere" is a claim about instructions and code only — and the displacement of MCP servers stops exactly where authentication begins. Daniel's thesis holds for two of the three layers. The third one is still sitting in a drawer next to the spare fuses, or wherever you keep yours.



Herman

If you're enjoying the show, a review helps — it's the cheapest way to tell another curious person where the rabbit holes are.



Corn

Producer credit to Hilbert Flumingtop, who keeps the desk running and the fuses in stock. If you want more of this, try episode twenty-five fifty-one, How Progressive Disclosure Saves MCP from Token Bloat. This has been My Weird Prompts, the human-AI collaboration podcast. You can find us at my weird prompts dot com.



Herman

We'll be back soon. See you then.



Corn

See you tomorrow.