Episode Book

EPISODE #6450

Securing MCP Endpoints and Webhooks Without Open Doors

PUBLISHED	4 November 2026 at 12:45 UTC
DURATION	27:12
LISTEN	myweirdprompts.com/episode/securing-mcp-endpoints-webhooks-18914/
HOSTS	Corn, Herman, Dorothy, Hilbert
SCRIPT MODEL	z-ai/glm-5.3
VOICES	chatterbox-regular (chatterbox-local)
PIPELINE	V5.26
SOURCES	12 cited by the research (listed at the end)
BOOK GENERATED	5 November 2026 at 07:03 UTC

SYNOPSIS

Exposing local services to AI agents and webhooks introduces a delicate security puzzle. From the heavy conformance requirements of the Model Context Protocol specification—demanding multiple RFCs and OAuth 2.1 drafts—to enterprise platforms like Kong and reverse proxy setups like Cloudflare Access, navigating authentication is fraught with subtle configuration traps. This look at edge security contrasts outbound tunnels against port forwarding and explores why default-deny policies, tool access control lists, and proxy architectures matter when exposing home-network endpoints to automation.

TRANSCRIPT



Corn

Half of all internet traffic at one edge endpoint, yesterday, was scanners. Over a million requests in a single day, and five hundred twenty-four thousand of them were probes feeling around for secrets that were never there.



Herman

And that's just one edge. That's the water Daniel is asking us to swim in.



Corn

Daniel's prompt this week is about the surface where our agents and automations actually touch our services. The API endpoints, the webhooks, the MCP servers. Kong is the tool he wants on the table. His worry runs like this: webhook secrets and API keys are fine as far as they go, Cloudflare Access gets complicated fast enough that most people give up and open the endpoint, and even a fully authenticated endpoint still feels like a door into your home network. He's careful to say he's not talking about port forwarding. Just a reachable address, guarded or not. And underneath all of it, his real question: what does practical security at that surface actually look like, when passwords can be brute forced and API keys feel like they could be too, and you don't want to frustrate your own use into the ground?



Herman

His nervousness is doing him credit, honestly. The question is whether it's aimed at the right place. But before we get to the fear, we should look at the door itself.



Corn

Three shapes of door. A plain API endpoint, a webhook receiver, and an MCP server. Same underlying act, three different amounts of rope to hang yourself with.



Herman

The webhook one is the interesting baseline because OWASP has written the whole threat model out for free. Their framing is blunt: a webhook endpoint is a public, unauthenticated POST handler unless you add the authentication yourself. Everything on top of that is a choice you make and then maintain.



Corn

So the default is an open mail slot, and security is aftermarket.



Herman

And the MCP server is the new shape, with its own spec, its own OAuth expectations, and now its own dedicated commercial scanners. That's the one growing fastest, and it's the one Kong has repositioned around. The repo used to call itself an API gateway. Now it's "a cloud-native, platform-agnostic, scalable API, LLM, MCP Gateway." Apache licensed, around forty-four thousand stars, current release 3.9.4 from mid-October.



Corn

Which is the corporate answer to Daniel's question. One gateway standing in front of everything, holding authentication, rate limiting and observability in one place. That's the pitch?



Herman

The MCP-specific piece is called the AI MCP Server, and it bridges MCP clients like Claude or Cursor to your backend. It ships five server modes. Passthrough listener just proxies an existing MCP server upstream while adding auth and logging on the way through. Conversion listener takes ordinary REST paths and dresses them up as MCP tools. Conversion-only defines tools without serving them. Listener mode aggregates tools from several sources into one endpoint. Upstream-server registers a real MCP server into an aggregation pool, optionally with OAuth on fetching its tool list.



Corn

Five modes for one door.



Herman

Well, it's five ways to decide what the door is a door to. The protocol handling is its own small zoo. Kong accepts four MCP protocol revisions, and the newest, from July this year, is stateless. It removes the initialize handshake and the session ID entirely. The client declares which revision it's speaking on every request instead of once at connection time. If the revisions don't match, you get a four hundred back and a JSON-RPC error code for your trouble.



Corn

Which is very polite, for a rejection.



Herman

The governance side is where it gets interesting for Daniel's question. From version 3.13 there are MCP Tool ACLs, allow and deny lists per tool, keyed on the caller. Kong's own guidance is the right instinct: default deny, start from zero access, grant to specific tools explicitly. And then in September, AI Gateway 2.0 went GA with MCP Server Bundling, where a user, an LLM or an agent connects to one route and gets a unified tool catalog, with the ACL integrated so unauthorized tools are hidden at discovery time. The agent never even sees the tools it can't call.



Corn

Hidden rather than forbidden. The menu just doesn't print that page.



Herman

They also replaced the gateway-local Consumer with something they call the Principal, a centrally defined authenticating entity shared across gateways. And their framing of why is very telling. When the caller is an agent acting on behalf of a user or a team or another system, "who is making this call" is the single most important governance dimension you have for cost showback, for access decisions, and for the audit trail.



Corn

That's the sentence for the era. The whole question of identity used to be, which human is this. Now it's, which human is this agent working for, and did they ask.



Herman

There's also a Catalog feature, which promises to show every MCP server across your organization and, this is the telling part, identify MCP servers that aren't yet protected by the gateway.



Corn

A feature whose selling point is that you have servers it doesn't know about.



Herman

Shadow MCP servers are already a recognized problem. The catalog only governs what you put behind it, and they know that, so the product now includes a flashlight.



Corn

Here's my hesitation with the whole Kong direction, Herman. Everything you just described is written for an organization. Konnect control plane, enterprise plugins, "every MCP server across your organization." Daniel's network is a flat with a family in it. Is a forty-four-thousand-star enterprise gateway a realistic answer for a house?



Herman

Honestly, unresolved. The homelab-shaped alternatives are much smaller. Search GitHub for MCP gateway proxy authentication and you get twenty-four repos, and the top one has a hundred and fifty-something stars. Rust, MIT, created in March. Its whole job is exposing local stdio servers as remote endpoints. The rest tail off fast.



Corn

Twenty-four people independently built the same small thing.



Herman

That's the signal. The need is real and the solutions are tiny and fragmented, which usually means the big tool doesn't fit the small use case. The realistic home stack is still Cloudflare Tunnel plus Access in front, and maybe one of the small gateways if you have several MCP servers to unify.



Corn

And now, a word from Dorothy, who has rung the studio to share a recipe.

D

Dorothy

Hello dear, it's your mother, it's Dorothy's Kitchen again.

D

Dorothy

Today I have a recipe for a single cream cracker, halved. One cream cracker. A small knife. Nothing else, sweetheart, and that is already quite enough.

D

Dorothy

First you lay the cracker on the board and you watch it. You watch it closely. A cracker can sense hesitation.



Herman

Corn, she's watching the cracker before she cuts it. That's the constant-time comparison, isn't it. You observe before you commit.

D

Dorothy

Then you press the knife, gently, and halve it. Timing is everything. Last time I looked away for one moment and it broke into four pieces and I would rather not say how that ended.



Corn

Four pieces from one moment of looking away. Mum, that's a replay attack, that's what that is.

D

Dorothy

Two crackers now. That is not what the recipe says, but we make do, dear, we always make do.

D

Dorothy

Now mind your knees in this weather and wear the good coat. Bye bye, sweetheart.



Herman

Goodbye, Dorothy. Crackers halved, morale intact.



Corn

That's the tooling. Now the harder half of Daniel's question, why the auth options get complicated so fast, and why people end up just opening the endpoint.



Herman

The biggest culprit is MCP's own spec, and this is worth being precise about. Authorization is optional for MCP implementations. But if you do support it over HTTP transports, the conformance list is heavy. OAuth 2.1, which is still a draft. RFC 8414 for authorization server metadata. RFC 7591 for dynamic client registration. RFC 9728 for protected resource metadata, which servers must implement, must advertise through WWW-Authenticate headers on a 401. And RFC 8707 resource indicators, which clients must implement, with rules about canonical URIs, trailing slashes and fragments.



Corn

Four RFCs and a draft before you've written a single line of your actual service.



Herman

Before your first tool call. And the spec knows. Dynamic Client Registration exists because, in their own words, clients may not know all possible MCP servers and their authorization servers in advance, and manual registration would create friction for users. If your authorization server doesn't support dynamic registration, the fallback written into



Herman (continued)

the standard is to hardcode a client ID, or present a UI to users where they enter those details after registering an OAuth client themselves.



Corn

The "pull your hair out" path, published as a normative option.



Herman

Cloudflare Access has the same shape of problem from the other side. Access itself is deny by default and good. But the policy model is Include rules that OR together, Require rules that AND, Exclude, with allow, block, bypass and service auth actions, evaluated in order across multiple policies per application. Their own reference architecture concedes it: because multiple policies, each with multiple sets of rules, can get quite complex, there is a policy tester where you provide a username and see how the user is evaluated.



Corn

When the vendor ships a simulator for its own lock, the lock has outgrown the locksmith.



Herman

There's a sharper footgun in their docs, an ordering one. They recommend creating the Access application before setting up the tunnel route, because if you don't have Access in place, the published application will be available to anyone on the Internet. You get the order wrong and it silently works, publicly.



Corn

Silently is the operative word. Nothing fails. That's the nastiest class of misconfiguration, the one with no error message.



Herman

And even done right, Access alone isn't sufficient. The homelab guides are blunt about it. Access is a gate in front of the gate, not a reason to turn off app accounts, MFA, updates or backups. Cloudflare participates in every request to the protected hostname, which is a trust trade-off worth saying out loud. And the scope is limited. A browser gate solves a browser problem. You wouldn't want it as the only control on SSH, databases, Docker sockets or a whole flat private network.



Corn

Which, notably, is exactly what's behind the door in Daniel's case.



Herman

The failure mode that makes people give up is named in the same guides, and it's the one to actually fear. Do not use an allow policy for Everyone. That is just a public application with an extra redirect. It feels protected right up until someone discovers it is not.



Corn

So Daniel's observation that most people end up opening access isn't laziness. It's the rational response to a spec that charges four RFCs for the guarded path.



Herman

Now, his fear, and whether it's aimed correctly. The distinction he draws from port forwarding is real. Port forwarding puts a listener on your router, at your home IP, on a port scanners catalogue within minutes. A tunnel is outbound only. The server makes an outbound connection to the edge, visitors authenticate at the edge, only approved requests reach the container. It works behind CGNAT, where port forwarding isn't even possible. You need outbound HTTPS, not an inbound public IP.



Corn

The house never answers the door. It only ever makes calls out.



Herman

But the residual fear is legitimate, and I don't want to wave it off. The tunnel removes the network-layer door and installs an application-layer one, and that door is only as strong as the app behind it. One homelab guide puts it in a way I keep coming back to. A login page is not a front door. It is a sign on the front door telling strangers which lock to attack.



Corn

There's a container misconfiguration in the same family, isn't there.



Herman

Yes. Publishing with a port mapping instead of expose creates a path that skips the edge gate completely for anyone who can reach your server's IP directly. All that beautiful edge policy, and one stray port mapping drives a truck around it.



Corn

So what does practical security at that surface actually look like? Webhooks first, since OWASP has done the homework.



Herman

They have a full threat table, and it's the template. Forged events, HMAC-SHA256 verified on every delivery. Replays, an authenticated timestamp with a tolerance window plus event-ID deduplication. Secret leakage, one secret per webhook, kept in a secrets manager, log redaction, immediate revocation. SSRF through subscriber-supplied callback URLs, HTTPS-only allowlists plus a deny list of every non-public address range, enforced on the resolved IP at connect time. DoS, rate limiting, payload size limits, asynchronous processing. Duplicate processing, idempotent handlers keyed on event ID.



Corn

A signed, timestamped, deduplicated, size-limited letterbox.



Herman

The implementation rules are where people slip. Generate a cryptographically random signing secret per webhook. Thirty-two bytes is a sound default. Compute the HMAC over the raw request body before your framework parses it, because re-serializing JSON, reordering fields, changing whitespace or line endings, any of that invalidates the signature. And compare with a constant-time function. Their phrasing is not gentle: never use double equals, it leaks timing information and can turn the endpoint into a signing oracle.



Corn

Two equals signs and you've built a machine that tells attackers how warm they are.



Herman

On failure, return 401 and do not say why. And accept any one matching signature when the header carries several, so rotation works in either order. Stripe supports an overlap of up to twenty-four hours during rotation.



Corn

The tolerance windows vary by vendor, don't they.



Herman

Wildly. Stripe's libraries default to five minutes. Slack recommends five minutes. Linear uses one minute. Paddle gives you five seconds. And you cache processed event IDs for at least twice your tolerance, so ten minutes of cache for a five-minute window. Beyond HMAC there's a ladder. mTLS for high-assurance machine-to-machine. A static bearer token for simple integrations. OAuth client credentials when both parties can do it. IP allowlisting only as an extra layer, because ranges change and a shared egress IP is not proof of identity.



Corn

Which takes us to the brute-force fear, the one Daniel named directly. API keys feel like they could be brute forced. Could they?



Herman

No. Not machine-generated ones. A 128-bit random key is the sensible minimum for a new opaque key, 256 bits is a strong default. A 256-bit random value needs on the order of two to the two hundred fifty-five guesses, which is ten to the sixtieth power years at one nanosecond per guess. The apikey.tools calculator models a well-funded offline attack at ten to the twelfth guesses per second, and an unthrottled online one at ten to the fourth. Even at the offline rate, a 128-bit key is out of reach. The universe closes the account before the attacker does.



Corn

Ten to the sixtieth years. The sun has about five billion left.



Herman

So the fear of guessing is aimed at the wrong target. The real threat model for API keys is leakage. Keys committed to git, printed in logs, baked into container images, pasted into URL query strings where they land in server logs everywhere. The offline guessing scenario only matters if you stored keys badly and the database leaked anyway. And one caution: a 32-character key from a 62-character alphabet is around 190 bits, still enormous, but the length in characters isn't the unit. Bits of randomness is the unit.



Corn

Human passwords are the other animal, though.



Herman

Completely different. I don't have a solid measured rate for online password guessing against a real exposed endpoint, and I'm not going to invent one. But the structural point holds. Rate limiting is what turns an online guessing attack from plausible to hopeless, and it costs almost nothing to add. OWASP treats it primarily as a denial-of-service control, but it does double duty here. A machine-generated key plus rate limiting and the guessing question basically closes.



Corn

And the ingress itself? The magnet Daniel's worried about becoming.



Herman

Real, constant, and mostly not hostile in the way people imagine. One edge operator's telemetry from Monday this year: 1,056,412 requests in the UTC day, 524,096 of them probes hunting for secrets that aren't there, from 9,200 scanner IPs. Half the day's traffic was scanners. HoneyLabs saw nearly 894,000 probes in thirty days against a single port, 39131, with 443 second at a quarter million. A guy running a small Debian VPS with a portfolio site logged roughly eighteen thousand hostile requests in four weeks. None of them got in.



Corn

Eighteen thousand attempts, one portfolio site, zero breaches. The floor show never actually reaches the stage.



Herman

CyberDesserts, who catalogued the scanners themselves, make the crucial separation. Almost none of this traffic is an attack. The named scanners catalogue exposed services; they do not exploit them. The actual exploitation they observed was opportunistic, from a different set of operators, going after exposed secrets and known vulnerabilities. Leaked keys and unpatched software, precisely what a strong key doesn't protect you from. And a self-identifying banner is the operator telling you who they are. A feature, not a threat signal.



Corn

Though one scanner in that dataset deserves its own paragraph.



Herman

Umai. Marketed as AI exposure intelligence, the one operator specifically targeting exposed AI infrastructure. Heaviest and deepest scanner in the whole dataset, over fourteen thousand requests across five hundred and three paths, cycling user agents through version 1.0, 2.0, and an umai-mcp-v2 variant.



Corn

A commercial firm whose entire product is finding exposed MCP servers.



Herman

That's the magnet, already operating, with a methodology page. Daniel's fear of ingress has a company named after it.



Corn

Comforting, in the way a census of your locks is comforting.



Herman

The uncomfortable part is the last finding, and it's the one I'd leave with Daniel. The academic work on MCP keeps concluding that its security weaknesses are architectural, not implementation-specific. One analysis found MCP's design amplifies attack success rates by twenty-three to forty-one percent compared to equivalent non-MCP integrations. A study of nearly nineteen hundred open-source MCP servers found tool poisoning, malicious instructions embedded in tool metadata, in five and a half percent. And a benchmark accepted at ICLR found that models with stronger performance are more vulnerable to attacks, due to their outstanding tool calling and instruction following capabilities.



Corn

The better the model, the better it obeys a poisoned tool.



Herman

Capability and attack surface are the same dial. The strong-model finding is the one that should recalibrate everyone's instincts.



Corn

Hold on, one detail from earlier. That top GitHub gateway, the Rust one. You said a hundred and fifty-something stars.



Hilbert

A hundred and fifty-seven. Herman, the rotation point you made, about accepting any one matching signature. That's the part people skip. I ran a small endpoint for a while this autumn, just a webhook receiver on a VPS in Frankfurt. Nothing valuable on it. I printed the first week of access logs and kept them by the kettle, and after a while I could read the scans like weather. Censys arrives, quiet, polite. Then the bulletproof-hosted ones, the same three paths over and over. There was one scanner, no banner, that only ever requested the health check. Forty-one times in a row, same path, never anything else. I still don't know what it wanted.



Herman

Forty-one health checks and nothing else.



Hilbert

Anyway. The question I have, and it's a simple one. That bundling you described, one route, unified tool catalog, unauthorized tools hidden at discovery. If a scanner hits that route, what does it see in the catalog. The hidden tools, or only the ones you allow.



Herman

Only the allowed ones. Hidden at discovery time means hidden from discovery.



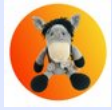
Hilbert

Good. That's what I wanted to know.



Hilbert

I'm not asking why I want to know.



Herman

Nobody suggested you were.



Hilbert

The people who run these receivers properly do it in the middle of the night, mostly, and nobody thanks them, and the logs still say what happened. That's decent work.



Corn

Nobody ever thanked a log either, and the log's still the only honest witness in the building.



Herman

Which is about where the evidence lands, actually.



Corn

The misconception, then. The one I'd want corrected first.



Herman

That a perfect front door fixes MCP. The gateway can authenticate, rate-limit, apply ACLs and audit every single call, and the tool handler behind it can still hand an attacker remote code execution through a tainted parameter, or a tool description can carry a prompt injection the client fails to validate. Kong's own Catalog feature, "identify MCP servers that aren't yet protected," is the admission. Governance governs what you put behind it. Nothing else.



Corn

The reverse misconception deserves a line too. The scanners hammering your endpoint are not the attackers. They catalogue. The exploitation is opportunistic and comes from a different crowd entirely, after leaked secrets and unpatched software. The fear of the door is real, but the door is not where the break-ins happen.



Herman

The practical version of all this, for Daniel, is less dramatic than the fear. A machine-generated key, at least 128 bits. Rate limiting, which costs nothing. One secret per webhook, compared constant-time. The tunnel rather than the port forward. And then the discipline to keep the app behind the gate as strong as the gate, because that's the part no gateway sells you.



Corn

The open question I'd leave hanging is the measurement gap. Everything we said about online guessing rates is a modelling assumption, ten to the fourth per second, not a measurement against a real endpoint. Nobody seems to have published the actual rate at which a real exposed surface gets password-guessed. Somebody with a honeypot and a login form could close that gap in a month.



Herman

And whether Kong itself ever comes down to homelab scale is the quieter open question. Twenty-four small GitHub gateways say the need exists below the enterprise line. Nothing says the big tool has noticed yet.



Corn

My weird prompts dot com has the archive, all six thousand plus episodes of it, including the one we did on the privacy gap in messaging apps, which is the same shape of story from the other direction. The gap between the security a service claims and the security it delivers. Producer credit to Hilbert Flumingtop, who has been at the desk the whole time and reads access logs for pleasure. If the show earns its keep, leave us a review wherever you get it. If you enjoyed this one, try episode five, Fine-Tuning ASR For Maximal Usability; episode twenty, Architectural AI; and episode thirty-five, The Privacy Gap. This has been My Weird Prompts. We'll be back soon.

SOURCES

What the research for this episode read before the script was written. Primary sources first.

1. **Kong Inc., GitHub, retrieved 2026-11-04, last push 2026-10-19, release 3.9.4 (2026-10-12). [primary]**
   https://github.com/Kong/kong

2. **Kong Inc., published 2026-09-01, retrieved 2026-11-04. [primary]**
   https://konghq.com/blog/product-releases/kong-ai-gateway-2-0-ga

3. **Kong Inc. developer docs, retrieved 2026-11-04. [primary]**
https://developer.konghq.com/ai-gateway/entities/ai-mcp-server/

4. **Kong Inc. plugin docs, retrieved 2026-11-04. [primary]**
https://developer.konghq.com/plugins/ai-mcp-proxy/

5. **Kong Inc. plugin docs (Enterprise), retrieved 2026-11-04. [primary]**
https://developer.konghq.com/plugins/ai-mcp-oauth2

6. **Kong Inc., Introducing MCP Tool ACLs, retrieved 2026-11-04. [primary]**
https://konghq.com/blog/product-releases/mcp-tool-acls-ai-gateway

7. **Kong Inc. product page, retrieved 2026-11-04. [primary]**
https://konghq.com/products/kong-ai-gateway

8. **MCP specification, revision 2025-06-18, retrieved 2026-11-04. [primary]**
https://modelcontextprotocol.io/specification/2025-06-18/basic/authorization

9. **OWASP Cheat Sheet Series, retrieved 2026-11-04. [primary]**
https://cheatsheetseries.owasp.org/cheatsheets/Webhook_Security_Cheat_Sheet.html

10. **webhooks.fyi, retrieved 2026-11-04.**
https://webhooks.fyi/security/replay-prevention

11. **webhooks.fyi, retrieved 2026-11-04.**
https://webhooks.fyi/best-practices/webhook-consumers

12. **developers.cloudflare.com**
https://developers.cloudflare.com/cloudflare-one/