Episode Book

EPISODE #6425

Mapping Undocumented APIs Without Breaking Them

PUBLISHED	3 November 2026 at 16:41 UTC
DURATION	28:37
LISTEN	myweirdprompts.com/episode/mapping-undocumented-apis-08153/
HOSTS	Corn, Herman, Hilbert
SCRIPT MODEL	z-ai/glm-5.3
VOICES	chatterbox-regular (chatterbox-local)
PIPELINE	V5.26
SOURCES	23 cited by the research (listed at the end)
BOOK GENERATED	4 November 2026 at 09:31 UTC

SYNOPSIS

API documentation is often treated as ground truth, but for builders constructing MCP servers, it is frequently outdated, incomplete, or aspirational. Probing what an API actually exposes requires a disciplined workflow rather than blind fuzzing. From generating OpenAPI specifications out of network captures using tools like mitmproxy2swagger to reading status code nuances and managing credential hygiene, this look into API mapping reveals how to reverse-engineer actual server behavior safely.

TRANSCRIPT



Corn

Daniel's builder instincts are showing again. He wants the whole workflow for mapping an API you're legitimately allowed to use, one where you've got a key, you're going through the front door, and the only problem is that the documentation is old, incomplete, or quietly fictional. And he draws a line first, which I appreciate: sniffing a web app's traffic to find hidden internal endpoints is, in his words, a Pandora's box. That's not what he's asking about. He's asking about the sanctioned case, end to end. How you probe what the API really exposes, how you document it, and how you turn that into a spec you or an agent can build against.



Herman

And the first thing worth doing is taking that distinction seriously, because it turns out the line doesn't sit where you'd expect. There's a line I keep thinking about from the builder plugin he maintains. The README says documentation is a hypothesis, the API is the source of truth. That's really the whole episode.



Corn

A hypothesis with a marketing budget.



Herman

It fits, though. And the reason this is a live problem right now rather than a niche hobby: MCP servers are being built for everything. There's one for a grocery list app, created end of last year, sitting at zero stars. Zero. That's the shape of the long tail. Thousands of tiny, unglamorous servers, each one built on top of some service's API that nobody documented properly, maintained by one person who mostly wanted their agent to manage their shopping.



Corn

Zero stars and still someone's production dependency.



Herman

Right, somewhere. Now, here's where Daniel's clean distinction gets complicated, and I want to be honest about it. His own builder plugin explicitly puts reverse-engineering an undocumented private API from browser traffic out of scope. It scopes itself to writing a new server for an API with no official MCP. But then the same playbook tells you to use the vendor's own web app as the map, because the web app calls everything, including what the docs omit.



Corn

So the technique is shared. The ethics live in what you do with the result.



Herman

Yes. And what you do with the result is: anything the docs don't mention goes into a separate pile, marked undocumented, and stays unexposed by default. Because undocumented behaviour can be withdrawn without notice, and a tool built on it will break silently.



Corn

So let's start with the case that makes this distinction load-bearing, because it inverts it. The Infomaniak story.



Herman

Infomaniak, the Swiss hosting provider. A builder made an unofficial MCP server for them, fifty-four tools, starting from their documented public API. Nice OpenAPI-ish spec, Bearer auth, everything you'd want. Then he tries to create a website. POST to the create-site endpoint returns 200 OK, with a site ID. The site never appears in the manager. No error. Just… nothing.



Corn

Two hundred with a receipt for something that doesn't exist.



Herman

So he does the diff. Opens the manager's network tab and compares what the manager itself is calling versus what the public API documents. And the manager isn't calling the public API at all. It's hitting a private proxy route under manager.infomaniak.com, authenticated with two session cookies instead of Bearer. Same pattern for databases, FTP users, mailboxes, password rotation. His conclusion was that the public API is read-mostly, and real automation requires the manager-private surface.



Corn

Which puts the builder in exactly the position Daniel's distinction is meant to avoid.



Herman

It does. And the honest move he made was a product decision, not a technical one. He recorded the private surface as undocumented and left it unexposed. He called the cookie-based manager auth a maintenance debt, because those cookies expire every few hours. So the sanctioned surface and the documented surface are not the same set, and the fact you can see the other one doesn't mean you should build on it.



Corn

Once you accept that, the question becomes how you find out what's actually there. Tooling.



Herman

The canonical tool is mitmproxy2swagger. It's been around, it's MIT, and it's had two big moments on Hacker News, one at nearly seven hundred points. What it does: you export a HAR file from your browser's DevTools while you click around the vendor's own web app, and it converts that captured traffic into an OpenAPI 3.0 spec. The workflow is deliberately two passes. The first pass emits every discovered path with an ignore prefix on it. You then hand-edit the YAML, strip ignore from the paths you actually want, fix up the path parameters, like turning inventory slash five slash categories slash thirty-one into a templated path with ID placeholders. Then the second pass generates the endpoint descriptions. It won't overwrite descriptions you've already written, and you can run it repeatedly against different captures and everything merges safely.



Corn

The ignore prefix is doing something quiet there. It's forcing you to say yes to each endpoint.



Herman

That's the design. The tool doesn't know what matters, so it makes you choose. There's a caveat worth flagging: the flags that include real request and response examples will happily bake your tokens and passwords into the schema. And the HAR files themselves hold cookies and tokens, so those never get committed anywhere.



Corn

A capture of your own session is a credential with a file extension.



Herman

Keep it in mind the whole time, yes. Now the diffing half is oasdiff. Once you've got an inferred spec from your captures, you diff it against the official one, and the diff is your list of documentation errors. That's the move. oasdiff grades every change: definite breaking, potentially breaking, and non-breaking. You can gate CI so a build fails on definite breaks, and there's a GitHub Action that annotates pull requests inline. There's a JVM alternative, openapi-diff, same idea.



Corn

And the newer ones?



Herman

Worth naming three. vespasian, from a pentesting firm, imports Burp captures, HARs, mitmproxy dumps, and outputs OpenAPI for REST plus GraphQL schemas and WSDL for SOAP. openapi-enrich has my favourite framing of the whole space: plenty of APIs have no specification, or one that stopped matching reality some time ago. What they have is traffic. And then tabapi, which is the most MCP-native of the lot. It drives a browser for you, clusters the requests into endpoint groups, infers the auth patterns, and exports an OpenAPI spec plus a TypeScript SDK plus MCP server config plus replay tests, all in one go. It auto-redacts tokens locally.



Corn

That's a lot of machinery for what used to be a Tuesday afternoon of clicking.



Herman

It is, and here's the negative finding, which I think is the interesting part. None of these tools knows or cares whether the API you're pointing them at is public or private. And none of them enforces any probing discipline whatsoever. You can point all of that machinery at an API and hammer it, and the tooling will cheerfully help. The discipline lives entirely in prose playbooks and in individual builders' habits.



Corn

Which is the part I actually want to talk about. What's the discipline?



Herman

Read before write. Reads can go unattended. Writes get confirmed one at a time, and you name what will change and where before you make the call. Sequential: one request at a time, no parallel bursts, no concurrency to speed up a survey. Bounded: a handful of calls per endpoint to establish shape and boundaries. Not fuzzing, not enumerating IDs, not scraping the dataset, not scanning for undocumented paths beyond what the docs hint at. The framing in the playbook is that this is ordinary client development, and the discipline exists because the account is real and the vendor's patience is finite.



Corn

Six things to establish per endpoint, I'm guessing, not sixty.



Herman

Six. Auth shape, confirmed with the cheapest read endpoint first. The real response shape from an actual payload, because docs examples are routinely abridged, stale, or hand-written. Which parameters are really required, by omitting each supposedly-required one once. Pagination: request page two, capture the cursor field names, find out how the last page signals it's last. Boundaries: the page-size max, one invalid value for the error shape, one bad-key call for the auth error shape. And types: string versus integer IDs, date format and timezone, nullability, the enum values actually returned.



Corn

My kind of list. Finite, and each item ends in a fact.



Herman

You'd have it laminated by Thursday.



Corn

I'd have it laminated before lunch.



Herman

The core skill underneath all of it is status-code reading, and this is where I think the craft actually lives. A 401 means the key is invalid, expired, or misplaced, and you check the header shape before blaming the key. A 403 means the endpoint exists but your key lacks the scope or your plan lacks the feature, which is materially different from a 404, and you record it. A 404 means the path is wrong or the resource is absent, and you have to distinguish the two before recording an endpoint as nonexistent.



Corn

How?



Herman

You calibrate. Three calls. A dummy ID on a known route. A known POST-only route hit with GET. And a completely made-up path. On many APIs a real route with a missing record says so in the body, a real POST-only route answers GET with a 405, and a nonexistent route gives a bare 404 identical to your made-up path. Once that pattern holds, you can send a GET to confirm whether a POST or PUT route exists without ever writing anything. The GET costs nothing. The write costs something.



Corn

So a 405 is the cheapest proof of existence there is.



Herman

In that setup, yes. Whereas a 400 that names a path parameter usually means your request got swallowed by a broader route with an ID placeholder, which is inconclusive. OPTIONS and CORS preflight are nearly useless, gateways often list every method for every path. The useful errors are 400 and 422, because the body usually names the real constraint, which is often not what the docs say. 429, you back off and record the observed limit. 5xx, retry once, then note it.



Corn

Herman, we did an episode ages ago on building speech recognition tools from scratch, and the same instinct showed up there: the manual is where you start, the machine is where the truth is. Same muscle, different machine.



Herman

Same muscle. Now, the hygiene rules, quickly, because they matter. The key comes from the environment, never a command-line argument, because that lands in shell history and process listings. Never a file in the repo, never a tool argument. You never echo it, you write Bearer and then stars. And you redact at capture time, not before committing. At the moment the file is written, so an interrupted session can't leave real data sitting on disk.



Corn

The stop conditions?



Herman

The terms of service prohibit it. A probe would write without confirmation. Rate limiting persists after you back off. The key turns out to have production access you didn't realise. Or the API diverges so far from the docs that the tool you planned no longer makes sense. You stop, and you reassess.
Ivan: Echo Tern. Seven four. Two zero nine. Seven four. The goose has left the yard and the eggs are warm. The ledger is open but the pages are blank. The gate was opened by the wrong key, so count the feathers twice. Nine one. Pipit.



Herman

Noted. So the capture side gives you a plausible spec, the discipline keeps you honest while you verify it.



Corn

What the hell was that.



Herman

The secret hygiene point people skip is the redaction timing. Before committing is too late.



Corn

Fine. Leaving whatever that was on the floor of the studio. So you've probed it, you've got an inferred spec, you've read the status codes. What do you actually write down?



Herman

This is the part I find elegant, so bear with me. The most useful artifact in this whole space isn't the OpenAPI file. It's a committed api-surface.yaml at the root of the repo, and it's not a spec, it's an evidence ledger. Every endpoint entry carries an evidence field: live, docs, or inferred. Plus a verified date, the auth required, the parameters with a documented default versus an observed default. Discrepancies between those two are, and I'm quoting the template here, the whole point of the file.



Corn

A spec tells you what the API does. That file tells you how you know.



Herman

That's the difference that matters. mitmproxy2swagger gives you a plausible OpenAPI file in two passes, but the file has no memory of how each fact was established. The evidence ledger is what survives contact with a drifting API. There's a docs block where each documentation URL carries a hash for drift detection, so you can tell later whether the



Herman (continued)

page you read has changed since you read it. There's a version signal block, like a response header that returns a version date, so you can detect the API moving even when the docs sit still.



Corn

And the six things you probed for become six columns.



Herman

They slot straight in. Pagination style per endpoint, error shapes, whether it's destructive, and then a smoke test block with a fixture prefix, so any records your tests create are identifiable and cleanable afterwards. The tools block is the MCP surface, and it's explicitly consolidated, not one tool per endpoint. One widgets tool with list, get, create, update, delete actions covering five endpoints. And there's an undocumented block, for the things probing found that the docs don't mention, exposed false by default. Plus a known discrepancies block, which is described as the maintenance backlog: doc says X, API does Y, first seen on such a date, action taken.



Corn

You realize what you're describing. It's a medical chart.



Herman

Go on.



Corn

Observations with dates, tests ordered, findings versus impressions, a plan that gets revised when the patient contradicts it. You spent thirty years writing those.



Herman

I did, and the resemblance had not escaped me. The chart is only useful because it remembers what the last person observed. What's your guess, though, on the three skills around this file. Create, revalidate, smoke test. Does the revalidate one actually get run, in the wild?



Herman

Honestly, I don't know, and that's the honest gap in the whole field. Nobody has measured how often unofficial MCP servers break when the underlying API drifts. The two best write-ups we have, Infomaniak and VWO, are single-case evidence. The revalidate skill exists precisely because nobody has quantified the failure rate. My suspicion is the honest answer is that most people ship, it breaks, and then they fix it, and the ledger is the difference between fixing it in an hour and starting over.



Corn

Which takes us to what actually goes wrong. Give me the VWO case, because that's the argument for the ledger, not just a war story.



Herman

VWO, the experimentation platform. An unofficial server, forty-five tools. Start with the documentation failures. The docs say campaign singular in the path. Singular 404s. Plural works. Custom widgets are served from the changesets path, not any form of widgets path. One endpoint's documentation points at a completely unrelated attribute-list endpoint, and the nav lists it as GET when the server had to infer a PATCH. The tool description literally tells the agent the endpoint is inferred, which I think is remarkable candour.



Corn

The tool confessing to its own uncertainty. More than most vendors manage.



Herman

Then the shape problems. List collections come back in two different shapes on the same endpoint depending on parameters. Campaigns with no status filter returns a wrapped object with a total count and an offset. Campaigns with a status filter returns a flat array. Labels and changesets omit the data field entirely when empty. A metadata field the docs



Herman (continued)

imply exists is never present on any of them.



Corn

Two shapes for the same endpoint is just cruelty.



Herman

Here's where it bit. An earlier version of the list-result helper handled only the flat-array case, and silently returned an empty array for the wrapped one. A workspace holding a hundred and forty-two campaigns reported count zero. Silent failure. The fix normalises every shape, and because the failure was silent, which is what made it dangerous, the fix also surfaces a warning telling the agent to report a bug rather than conclude there's no data.



Corn

The status parameter. Undocumented?



Herman

A real query parameter VWO does not document. Uppercase only, lowercase gets you a 400. And the valid values were harvested from the error message for an invalid one. Seven values. And omitting it returns soft-deleted campaigns too, which is its own trap.



Corn

The error message as the only accurate documentation. That's grim even by this episode's standards.



Herman

It gets better. Limit is silently capped at twenty-five on campaigns. You ask for a hundred, you get twenty-five, and the response looks like a final page because the next-offset logic keys off whether the page came back full. So a hundred-item request returns twenty-five items and the client concludes it's done. After capping the schema at twenty-five, paging



Herman (continued)

walks all hundred and forty-two campaigns across six pages.



Corn

So the API lies about completeness and the pagination agrees with the lie.



Herman

Coordinated. The writes, meanwhile, require a plural wrapper: the campaign object nested inside a campaigns key. Unwrapped gets you a 400 saying the request is not in desired format. And the asymmetry trap: reading a variation returns this internal operation-stack structure, and writing that same structure back is rejected. The write format is a raw changes string that VWO compiles itself. Read format and write format for the same object are different.



Corn

You mentioned the Infomaniak server had lessons of its own.



Herman

The transferable ones. Pagination must be invisible to the agent, because letting the model do pagination means it forgets, gets the first page, and reasons over incomplete data. Some Infomaniak endpoints page at twenty-five, others at fifty, and the tool always pages through everything and returns the merged list. Error shapes normalised, because the public API returns one shape and the private surface returns that, or a list, or raw HTML on auth failure. Logs go to stderr, not stdout, because stdio transport mixes the protocol with anything you print, so a stray log line corrupts the protocol.



Corn

The confirmation dance?



Herman

MCP has destructive-operation annotations, but they're hints, they don't enforce anything. The Infomaniak builder wrote a wrapper where the first call to a destructive tool returns a token with a sixty-second expiry and a plain-language what-will-happen string, and only the second call within that window actually deletes. He says it saved him from production accidents twice during dogfooding. His other line I love: Claude is enthusiastic. Give it a tool called delete site and a thread of context saying let's clean up old test sites, and it will happily delete production.



Corn

An agent with your key is you, minus your hesitation.



Herman

He also surfaces a locked flag on hosting listings, because if he hid it, the model would happily try operations on locked hostings and fail. Verbose output is fine, surprise failures aren't. And rate limits: sixty requests a minute sounds generous until a workflow iterates over fifty domains making three calls each, and you hit the limit in thirty seconds.



Corn

So what does turning findings into a spec actually cost. The conversion itself.



Herman

The standard algorithm is one MCP tool per path and method combination. Name from the operation ID, or synthesised from the path if there isn't one. Description from the summary, then the description, then a stub. Input schema merges path params, query params, and body into one JSON Schema. Output schema is the first successful JSON response only. The lossy parts are where it hurts. Pagination is the fundamental mismatch, REST paginates and MCP tools return single results. Binary uploads often need a hand-coded tool. Error schemas are never exposed, so the agent sees only the success shape and handles errors generically.



Corn

And the descriptions.



Herman

The sleeper issue. Auto-generated descriptions are syntactically valid and semantically miserable. Get users user id repos is what a synthesizer produces. Agents make worse tool selections from descriptions like that. They pick the wrong tool, fill in wrong arguments, or skip the tool when it was the right one.



Corn

So the generated spec gets you the scaffolding and loses you the judgement.



Herman

Which is exactly the work a generated spec can't do for you. The consolidation move and the hand-written descriptions are the human's remaining job. TrueFoundry's worked example: forty-seven endpoints, and hand-coding a proper toolbox took roughly a week per tool, at which rate forty-seven endpoints would be most of a quarter of engineering time.



Hilbert

It's the quiet failures, not the loud ones. Herman's got the villain backwards.



Corn

Quiet how?



Hilbert

The VWO thing, the count zero. You've all been saying silent failure is the danger. No. The danger is the confident one. Two hundred OK, a plausible ID, a clean response. The thing that tells you it saved when it didn't. I've been done over by that once and I've never trusted a success message since. Corn, you were saying earlier the API lies by omission. That's the one I came out about. My nephew does integration work, been at it fifteen years. He keeps a notebook. Paper. Every call he's ever made written in by hand, date, what he sent, what came back. The docs lie and the notebook doesn't, he says. Two columns. What the docs said, and what actually happened. He fills in both every time, even when they match. Corn, I've levelled your mic down twice while you were on the VWO numbers. The matching ones are the ones you stop checking. I've started one myself, these past six weeks. Not for API calls. It's not for anything you'd ask about, so don't. Nephew number two, that's the connection. Rodney, on their mother's side. He does the



Hilbert (continued)

integration work.



Herman

The matching entries are the ones that become assumptions, and the assumptions are what break.



Hilbert

Rodney says the same thing in fewer words.



Herman

The Infomaniak builder's confirmation token, the sixty-second window, that's the same instinct on the machine side. It assumes the first confident response is a lie until a human repeats it.



Hilbert

Sixty seconds is generous. Rodney gives himself ten.



Corn

Which leaves a couple of things unresolved. The biggest one, for me: everything in this episode that made the sanctioned workflow safe, the read-before-write, the bounded probing, the evidence ledger, all of it lives in prose playbooks and personal habits. No tool enforces any of it. And I don't know whether that's a gap waiting for a tool or whether it's the kind of thing that can't be tooled, because at bottom it's a judgement call about someone else's server.



Herman

I lean toward can't-be-tooled, myself. A tool can diff specs. It can't decide whether you're a guest or an intruder. That's a fact about your relationship with the vendor, and software is bad at relationships.



Corn

The other open question is whether anyone will ever measure the breakage rate. Whether unofficial servers breaking under API drift is a trickle or a constant background hum. My guess is constant, and invisible, because nobody files a bug when their zero-star grocery list server stops working. They just fix it or they walk away.



Herman

And if someone did measure it, I'm not sure it would change behaviour. Builders keep shipping and fixing. The measurement would mostly tell the rest of us how big the iceberg is.



Corn

The one thing I'd take from this: sanctioned and documented are different sets, and the gap between them is where all the work lives. The evidence ledger, the discrepancy column, the notebook. However you keep it, the artifact that matters is the one that remembers how you know.



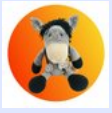
Herman

And the closing thought, from the REWE thread. That grocery CLI in Haskell, two hundred points on Hacker News, and a REWE employee shows up in the comments saying they've flagged internal interest in an official MCP server. The unofficial one created pressure for the official one. And the employee's own blocker is discovery: there's no established way of where an MCP server should live so agents can find it. That's the unresolved thing. Not the mapping, not the discipline. Where the map gets published.



Corn

If you take one thing from this episode: the API is the source of truth, but the truth needs a ledger.



Herman

Which, Corn, brings us around to what your nephew's notebook was doing before any of this had a file format.



Corn

Producers first, standards later. Thanks as always to Hilbert Flumingtop at the desk. For more along these lines, there's episode sixteen, On Deepfakes, SynthID, And AI Watermarking; and episode thirty-five, The Privacy Gap. This has been My Weird Prompts, the human-AI collaboration podcast.



Herman

If the show's earned it, a review on Spotify goes a long way. You can also find us at my weird prompts dot com.



Corn

See you tomorrow, brother.



Herman

See you tomorrow.

SOURCES

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

1. **README, v0.15.0, MIT [primary]**
https://github.com/alufers/mitmproxy2swagger

2. **README [primary]**
https://github.com/nextafter-michael/unofficial-vwo-mcp-server

3. **README [primary]**
https://github.com/danielrosehill/Unofficial-MCP-Builder-Plugin

4. **probing playbook [primary]**
https://github.com/danielrosehill/Unofficial-MCP-Builder-Plugin/blob/main/reference/probing-playbook.md

5. **template [primary]**
https://github.com/danielrosehill/Unofficial-MCP-Builder-Plugin/blob/main/templates/api-surface.yaml

6. **README, MIT, alpha [primary]**
https://github.com/Lay4U/tabapi

7. **README [primary]**
https://github.com/oasdiff/oasdiff

8. **breaking-change docs [primary]**
https://github.com/oasdiff/oasdiff/blob/main/docs/BREAKING-CHANGES.md

9. **oasdiff GitHub Action docs [primary]**
https://www.oasdiff.com/docs/github-action

10. **HN, Show HN: CLI to order groceries via reverse-engineered REWE API (Haskell), 2026-03-30**
https://news.ycombinator.com/item?id=47571183

11. **HN, mitmproxy2swagger Show HN, 2022-05-12 (headline only)**
https://news.ycombinator.com/item?id=31354130

12. **HN, mitmproxy2swagger repost, 2025-01-02 (headline only)**
https://news.ycombinator.com/item?id=42572662

13. **DEV Community, Infomaniak MCP build write-up**
https://dev.to/mogacode/building-an-mcp-server-for-a-swiss-hosting-provider-and-what-reverse-engineering-its-manager-2fg9

14. **TrueFoundry, OpenAPIMCP conversion guide**
https://www.truefoundry.com/blog/openapi-to-mcp-server-conversion

15. **Apidog, OpenAPI diff in CI (openapi-diff comparison)**
https://apidog.com/blog/openapi-diff/

16. **README (search snippet)**
https://github.com/praetorian-inc/vespasian

17. **README (search snippet)**
https://github.com/MarkRosemaker/openapi-enrich

18. **README (search snippet)**
https://github.com/dolejska-daniel/uexcorp-openapi

19. **v0.4.0 (search snippet)**
https://pypi.org/project/openapi-agent-ready/

20. **README (search snippet)**
https://github.com/sgaluza/api-to-mcp

21. **README (search snippet)**
https://github.com/adityaaa-IIT-BHU/keymaker

22. **NotePM MCP write-up, 2026-01-03 (search snippet)**
https://nozomi.bike/articles/2026-01-03-notepm-mcp-server/

23. **GitHub search result, created 2025-12-27**
https://github.com/mplabs/anylist-mcp