TYGART MEDIA · NOTES FROM THE BOT LAB

An AI Has Knocked on This Endpoint 3,402 Times. Nobody Answered.

By William Tygart · November 2026

In 21.7 days of Apache logs from one of my own sites, a single path took 3,402 hits. Every one of them came in with the user agent `Claude-User`. Every one of them went out with an HTTP 404.

Not 98% failure. Not a flaky endpoint. One hundred percent, 3,402 for 3,402.

The path is `/wp-json/claude-mcp/v1/mcp`.

If you build agent integrations, you already know what that path is trying to be. For everyone else, two sentences: MCP — the Model Context Protocol — is a standard way for an AI model to connect to external tools and data sources, so the model can do more than chat; it can query, retrieve, and transact. An MCP endpoint on a website is a door built for machines: a place where an AI agent shows up, identifies itself, and asks to *use* the site, not just read it.

Somebody's Claude configuration points at my site as a tool. And the door it's knocking on doesn't exist.

The irony is the finding

The site is theuniversalcommerceprotocol.com. It exists to cover the Universal Commerce Protocol — a standard for machines transacting with businesses. Its entire subject is agent commerce.

Here is how the machine audience actually behaves on it, from the same log window: 223,953 page requests total, 8,154 of them from AI crawlers and agents. Of those 8,154 AI hits, 3,332 — 40.9% — went to specification pages. The crawlers are also probing `/.well-known/ucp` paths: the discovery locations where a machine goes to ask a site, formally, *what can I call here?*

So the machine audience on this site is doing exactly what the protocol says machines will do. It reads the specs. It checks the well-known discovery paths. And then — 3,402 times — it knocks on an MCP endpoint that somebody configured, presumably because they expected this site, of all sites, to have one.

It doesn't. WordPress has no route registered under that namespace, so the REST router does the only thing it can do: 404, 3,402 times in a row, silently.

No error dashboard lit up. No user complained. Nothing in any analytics tool I run would ever have shown me this, because failed API calls to a nonexistent endpoint don't render pages, don't fire tags, and don't show up as sessions. The only place this demand exists is the origin server log. If I didn't read logs, I would not know an AI has been trying to integrate with my site for three weeks.

What I don't know

I want to be precise about the limits of this finding, because the unknown is part of it.

I don't know who configured the integration. The log gives me a user agent and a count, not an identity or an intent. The honest possibilities, in no particular order:

- Developers building or testing UCP implementations, pointing Claude at the most obvious endpoint shape on the most obvious domain.
- Experimenters working through MCP tutorials or examples, using a plausible-looking endpoint on a protocol site.
- Some directory, registry, or example config somewhere that lists this endpoint — an endpoint that was never stood up — and everyone who copies that config knocks on the same nonexistent door.

Any of those could be true. Several could be true at once. What the log establishes is narrower and, I think, more interesting than any of the explanations: the *demand is real, repeated, and structured*. Random scanners don't hit one REST namespace 3,402 times with a consistent agent user agent. Something out there believes this endpoint should exist, and keeps acting on that belief.

I also can't tell you what a successful call would have been worth — what the agent would have queried, listed, or transacted if the door had opened. That's the nature of a silent failure. The cost isn't measurable from my side of the log. It is exactly the absence that's the data point.

Agents are clients now

Here's the shift I think this exposes, and it's the reason I'm publishing a log line instead of just fixing a route.

We have spent two years learning to think of AI systems as *readers*: crawlers that fetch pages, retrievers that quote them, assistants that cite them. That framing produced a whole discipline — make your pages findable, readable, citable. I do that work; it's most of what this lab does.

But a reader doesn't knock on an MCP endpoint. A *client* does. A client arrives with a configuration, an expectation of an interface, and a task it intends to complete against your infrastructure. When the interface isn't there, the client doesn't leave a comment or send an email. It logs a 404 on its side, maybe retries, maybe moves on — and you never hear about it unless you read your own logs at the level of individual paths and status codes.

A 404ing agent endpoint is the new dead API. Dead APIs used to be a developer-experience embarrassment you found in your own monitoring, because your own integrators would complain. Agent endpoints fail to an audience that never complains. The integration attempts are the demand signal, the failures are total, and the only witness is a log file most operators never open.

The diagnostic question I'd put to every technical operator is the one this forced on me: **what are agents trying to call on your stack, and what do they get?** You cannot answer it from a dashboard. You answer it by grouping your log by path, user agent, and status code, and looking specifically at the machine-shaped requests — the REST namespaces, the well-known paths, the protocol endpoints — that return 404. Everybody audits their 404s for broken *human* links. Almost nobody audits them for broken *machine contracts*.

On my site, that audit took one query. The answer was 3,402 knocks, zero answers.

Where this stands

Diagnosis is where this ends, for now. I've deliberately not stood up a replacement endpoint just to make the number go away — I don't yet know who's knocking or what they'd want from the door, and building the wrong interface to satisfy an unknown client is its own kind of failure. The finding is published first, because the pattern matters more than my one endpoint: if it's happening on a protocol site whose operator reads logs for a living, it's happening everywhere, unnoticed.

Check your logs. Look for the knocks.

We do AI citation restoration. When the AI can't find you, can't read you, or keeps citing somebody else — or keeps knocking on a door you never built — send us the worst one you have. Your RFPs, your disasters, the problem everyone else already failed to fix. We come from the restoration industry. We know what's on the floor, and cleaning it up is the job. And if you're a developer who just needs help on the hard part, get a hold of us.