Episode Book

EPISODE #5861

Inside Cloudflare's Bot Check: TLS, JA4, and the Sand-Timer

PUBLISHED	11 October 2026 at 11:02 UTC
DURATION	30:01
LISTEN	myweirdprompts.com/episode/cloudflare-bot-detection-explained-09876/
HOSTS	Corn, Herman, Hilbert
SCRIPT MODEL	deepseek-flash
VOICES	chatterbox-regular (chatterbox-local)
PIPELINE	V5.3
SOURCES	15 cited by the research (listed at the end)
BOOK GENERATED	19 October 2026 at 17:40 UTC

SYNOPSIS

That grey page with the spinning circle — "Checking your browser before accessing" — hides a two-layer detection system that has usually made up its mind before the page even renders. This episode walks through how Cloudflare's anti-bot machinery actually works: the TLS handshake fingerprints (JA3, and how Chrome 110 broke it, and how JA4 fixed it), the HTTP/2 SETTINGS-frame fingerprint Akamai figured out in 2017, the 1-to-99 bot score, and the silent JavaScript battery of proof-of-work, proof-of-space, and web API probes that runs afterward. The throughline: the strongest signal isn't any single fingerprint, it's the contradictions between them — a real browser's TLS, HTTP/2, and JavaScript layers all come out of one binary and agree by construction. Bots borrow each layer from a different library, and the seams show.

TRANSCRIPT



Corn

You know the shape of it. Grey page, white spinner, "Checking your browser before accessing." Two seconds of nothing. Then the site you actually wanted.



Herman

And if you're like most people, you assume those two seconds are the check.



Corn

Right. So Daniel's been running Cloudflare on his own sites. He's flipped on the strict anti-bot settings, he's used Under Attack Mode, and he's realised he has no idea what the machine is actually doing while his visitors stare at a sand-timer.



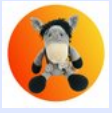
Herman

Which is a fair thing to want to know. You're paying for a security product and you can't see inside it.



Corn

So he's asking for the under-the-hood account. What signals get read, how the thing adjudicates in a couple of seconds, what those dashboard toggles actually change, and why it sometimes locks out a real person. He puts it well. He says he doesn't know what's happening "at the web firewall level" when it decides whether you're a genuine visitor or a bot.



Herman

The short answer is that by the time you see the spinner, most of the decision has already been made.



Corn

And that's the part worth unpacking. Because the visible wait, the bit you experience, is not the bit doing the work.



Herman

Here's the frame I want to put on it, and it holds for the whole episode. There are two layers. There's a network layer that forms a verdict before a single byte of JavaScript runs. And there's a JavaScript layer that runs on top of a network that has already made up its mind.



Corn

So the sand-timer is the second layer showing its face.



Herman

Partly. Let's start with the box the visitor actually receives. What gets served in place of the page you asked for is an HTML document. It carries JavaScript, and that script talks back to Cloudflare's challenge platform under a path that starts slash cdn dash cgi slash challenge dash platform. If the visitor passes, the browser is handed a cookie called cf_clearance, and it gets bounced to the destination it originally wanted.



Corn

So the cookie is the hall pass.



Herman

It's the hall pass, and it's stamped with which challenge gave it to you. We'll get there, because that stamping matters. But Cloudflare's own documentation describes the challenge as a series of checks evaluating client-side signals, data gathered from the visitor's browser environment. And it says, plainly, that most visitors will pass challenges automatically without interaction.



Corn

Which is the experience Daniel's describing. No action required.



Herman

But a lot happening. The thing I keep coming back to is the spin, the visible sand-timer, is partly theatre and partly a genuine cost being imposed. And the detection, the actual judging, is mostly invisible.



Corn

So the page is the announcement of a verdict, not the deliberation.



Herman

The deliberation happened earlier. Let's start where the decision actually starts, which is the handshake, before a single byte of JavaScript.



Corn

Go on.



Herman

Cloudflare terminates TLS at its edge. That means the secure connection ends at their machine, not at Daniel's server. And the first thing that arrives in that handshake is the ClientHello, which is sent in the clear by design. You can't encrypt the negotiation of encryption. So before Cloudflare has decrypted a single byte of HTTP, it has already got a full picture of how this client negotiates.



Corn

What's in a ClientHello that's worth reading?



Herman

The cipher suites the client offers, the TLS extensions, the order it lists them in, the elliptic curves it supports, the signature algorithms. And the key insight, the one that made this a discipline, is that these are not standardised into a single correct order. A real browser from a single vendor sends them in a consistent sequence, because it's one piece of software with one code path. So the order itself is a signature.



Corn

And someone gave that signature a name.



Herman

JA3. Salesforce, 2017. You take five fields out of the ClientHello, you hash them with MD5 in the order they were sent, and you get a fingerprint. It was a clever bit of work and it had a good long run. Then Chrome 110, in January 2023, started permuting the order of TLS extensions on a per-connection basis.



Corn

Deliberately?



Herman

Deliberately. It broke JA3 for anyone relying on extension order, which was everyone. And that's what motivated JA4. FoxIO, late 2023. The fix is elegant. Instead of hashing the fields in the order they arrived, JA4 sorts the cipher list and the extension list into hex order first, then hashes. So Chrome can shuffle all it likes, the fingerprint comes out the same.



Corn

So JA4 is Chrome's shuffle, neutralised.



Herman

Neutralised. And the format it spits out is three parts separated by underscores. If I gave you a real one, you'd see something like t thirteen d fifteen sixteen h two, then an underscore, then a twelve-character hex string, then another underscore, then another hex string.



Corn

That reads like a licence plate.



Herman

It functions like one. And the matching happens up at Cloudflare before it's decrypted anything. That's the first layer.



Corn

Is TLS the only thing they can read that early?



Herman

No, and this is the part I find impressive. There's a second fingerprint from the HTTP/2 layer. And this one was worked out by Akamai, presented at Black Hat Europe back in 2017. When an HTTP/2 client connects, it sends a SETTINGS frame, and it sends it with specific values in a specific order. It also sends a WINDOW_UPDATE with an increment, sometimes PRIORITY frames, and it orders the pseudo-headers on its requests in a fixed way.



Corn

And Chrome and Firefox do that differently.



Herman

Chrome sends its pseudo-headers in the order method, authority, scheme, path. Firefox sends method, path, authority, scheme. So if a request arrives claiming to be Chrome in its User-Agent header, but its pseudo-headers arrive in Firefox order, that client has



Herman (continued)

contradicted itself on the wire.



Corn

It told you two different stories in the same conversation.



Herman

And both stories are checkable. Now here's where it becomes hard to fake. A real browser's TLS layer, its HTTP/2 layer, and its JavaScript layer all come out of one binary. They agree by construction. They have to. An assembled bot, a scraper built out of libraries, borrows each layer from a different source. Maybe a Go TLS stack, a Python HTTP/2 library, and a JavaScript engine bolted on. And the contradictions between those borrowed layers are what the cross-layer check reads.



Corn

So the strongest signal isn't any one fingerprint. It's the disagreement between them.



Herman

That's the whole game. Consistency. Forging any single fingerprint is straightforward. Forging TLS, HTTP/2, pseudo-header order, JavaScript surface, and hourly global behaviour, all in agreement with each other, is close to the work of actually running the browser.



Corn

Which is the point. They're not trying to make it impossible. They're trying to make it not worth it.



Herman

Exactly that. And that's before we get to the score.



Corn

The score?



Herman

Cloudflare computes an integer between one and ninety-nine for a request. One means quite certain automated. Ninety-nine means quite certain human. And the buckets are rough. One is automated, two to twenty-nine is likely automated, thirty to ninety-nine is likely human.



Corn

So there's a wide uncertain band in the middle.



Herman

There's a very wide uncertain band, and that band is where the interesting decisions get made. Because a request scoring ninety-nine doesn't need much of a challenge. A request scoring five doesn't get a challenge, it gets blocked. It's the request sitting at forty that gets the silent battery.



Corn

The ambiguous ones are the ones who pay.



Herman

The ambiguous ones pay. And in 2024 they added something on top of this called JA4 Signals. The idea is that a fingerprint on its own tells you what the client claims to be. But if you watch a fingerprint across the whole internet for an hour, you learn what it actually behaves like. So they compute behavioural metrics per fingerprint over the last hour of global traffic. How often does this fingerprint show up on real browsers. Is it arriving on HTTP/2 or HTTP/3.



Corn

And the scale of that.



Herman

Over fifteen million unique JA4 fingerprints, generated from more than five hundred million user agents, across billions of IP addresses, every day. So a brand new fingerprint showing up for the first time, claiming to be a browser nobody has ever seen, is itself a signal.



Corn

Because no human is running a browser nobody else on earth is running.



Herman

Not in those numbers. So that's the network layer. It's decided. Now the JavaScript runs, and this is the battery.



Corn

The silent part.



Herman

Cloudflare's own description of it, from their 2022 launch, is a series of small non-interactive JavaScript challenges. Proof-of-work, proof-of-space, probing for web APIs, and various other challenges for detecting browser quirks and human behaviour.



Corn

So there are four or five distinct mechanisms stacked.



Herman

Stacked, and running together. Proof-of-work is the one people talk about most. The script takes a hash function, SHA-256, and iterates it. It's looking for a nonce that produces a hash with a certain number of leading zero bits. That's the difficulty knob. And it's adaptive. For a suspicious client it might be set to roughly twenty bits, which is about a million iterations.



Corn

A million hashes.



Herman

In a real browser on real hardware, with the GPU available, that's fifty to five hundred milliseconds. You'd never notice it. Headless, without GPU acceleration, the same puzzle takes five to ten seconds.



Corn

And that's the sand-timer.



Herman

That's a big chunk of the sand-timer, yes. And here's why it's clever rather than just annoying. A single human browser never notices the cost. A bot farm pays it a million times over. You're not detecting the bot, you're changing its economics.



Corn

You're not building a wall, you're setting a toll.



Herman

That's the right shape for it. And then there's proof-of-space, which is the companion piece. Instead of a CPU-bound puzzle, it's a memory-bound one. The client has to allocate and touch a chunk of memory in a way that's hard to parallelise cheaply. The point is that scraper hardware is usually optimised for doing lots of cheap CPU work in parallel, and



Herman (continued)

memory allocation is a different bottleneck.



Corn

So if proof-of-work doesn't slow them down, proof-of-space might.



Herman

Different resource, different constraint. And then the web API probes. These are the ones that read the browser environment directly. The script checks navigator dot webdriver, which is a flag that shows up when a browser is being driven by automation tooling. It checks navigator dot plugins. It checks for window dot chrome, which is present in real Chrome but frequently absent in things pretending to be Chrome. It checks Notification dot permission, it checks the WebGL rendering context, it checks the Canvas pipeline.



Corn

And the timing.



Herman

And the timing, which I think is the most underrated one. It reads performance dot now, repeatedly, and looks at the consistency of the intervals. A real browser running on real hardware has jitter and drift that come from the operating system scheduler and real human interaction. A headless browser running a tight automated loop has timing that is too regular, or too fast, or has impossible precision.



Corn

A machine that never breathes.



Herman

And one specific probe worth naming. WebGL has an extension that exposes the renderer string. On real hardware that string names a GPU. On a headless browser it frequently comes back with one of a couple of software renderers, and seeing one of those is an



Herman (continued)

immediate red flag.



Corn

Because you're claiming to be a laptop and your graphics card is a piece of software.



Herman

And then the fingerprints themselves, which people often conflate with the probes. Canvas, WebGL and audio-context fingerprints get computed and hashed into a single payload, which is sent to Cloudflare's edge. The canvas one is the classic. You draw text and shapes to a canvas and read back the pixel data. Tiny differences in font rendering, anti-aliasing, subpixel handling across machines mean two machines produce slightly different images, and the hash of that image is a fairly stable identifier.



Corn

So it's an identifier as well as a check.



Herman

It's both. It's an identity signal and a consistency signal. And speaking of identity, there's the obfuscation layer around all of this. Independent research on the challenge script describes it as obfuscated with a custom version of a widely used JavaScript obfuscator, and then it loads and executes a small custom virtual machine, which runs what the researchers call sub-challenges. So the code that does the fingerprinting isn't running as normal JavaScript in your browser. It's running inside an interpreter that the script brought with it.



Corn

Why bother?



Herman

Because you can't easily read, debug, or patch a program that only exists as bytecode for a virtual machine that was written for one page load.



Corn

So if you want to know what they're checking, you have to reverse-engineer their interpreter first.



Herman

Every single time it changes. And that connects to the part nobody outside Cloudflare gets to see. The exact contents of that payload, the field list, the byte layout of the token, the weighting between signals, the specific APIs probed on any given day, none of that is public.



Corn

Hm.



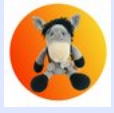
Herman

Deliberately. And anyone claiming to have the definitive field list is describing a snapshot of something built to change.



Corn

So how long does all of this take, from the user's side?



Herman

Cloudflare's own documentation says the "Checking your browser" challenge determines whether to block or allow a visitor within five seconds. That's the budget. And within that budget the network layer has already voted, the JavaScript battery runs, the payload goes back, and the verdict comes down.



Corn

Five seconds and a lot of work.



Herman

And then you get the page, or you get the spinner again, which is where this goes wrong for people. And that gets us to the dashboard, and what Daniel's toggles actually do.



Corn

The knobs.



Herman

Start with the one with the scariest name. Under Attack Mode. It sounds like a separate product, like you've flipped a switch and deployed a different system. It isn't. Under Attack Mode is a security level.



Corn

So it's a dial that was already there, turned to its highest position.



Herman

Turned all the way up. Cloudflare's docs describe it as performing additional security checks to help mitigate layer seven DDoS attacks, and it presents a Managed Challenge page to visitors. And the docs are honest about the cost. It's designed to be used as one of the last resorts. It will temporarily pause access to your site and impact your site analytics.



Corn

It pauses the analytics because it's pausing the visitors.



Herman

It's challenging the whole internet, so your analytics see a wall of challenge pages instead of pageviews. And the sensible way to run it is scoped. You can turn it on for specific paths, specific ASNs, specific countries, specific IP ranges. So you protect the login endpoint and leave the blog alone.



Corn

And Daniel mentioned enabling strict anti-bot settings. Is that the same thing?



Herman

It's the adjacent dial. Under Attack Mode is the blunt instrument. The bot-fight settings are the fine-grained rules, where you set the security level and the bot rules that apply to specific traffic. Both of them feed into the same challenge machinery, which is the part worth understanding. And the machinery has four types of challenge, and only one of them is what you actually want.



Corn

Go through them.



Herman

Managed Challenge is the first. It's the only one Cloudflare recommends. What it does is run the silent battery first, and escalate to an interaction only when the signals stay ambiguous. So it's conditional. You only get asked to do something if the machine isn't sure.



Corn

And that's the one that changed the numbers.



Herman

It changed the numbers in a way that's worth sitting with. When they rolled it out in 2022 they reported a ninety-one percent reduction in CAPTCHAs served, and average challenge time dropping from thirty-two seconds to about one second. And human visitors were thirty-one percent less likely to abandon a managed challenge than a CAPTCHA.



Corn

Thirty-two seconds down to one.



Herman

That's the whole argument in one number. The old experience was a puzzle and a wait. The new one is often neither.



Corn

Then there's the JavaScript challenge.



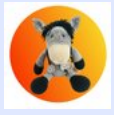
Herman

The JS challenge, the non-interactive one, is the oldest surviving type. And it's different in an important way. Managed Challenge is adaptive, it looks at how suspicious you are. The JS challenge isn't. It's a fixed battery that filters on capability rather than suspicion strength.



Corn

Meaning?



Herman

Meaning it's binary. A real browser sails through it. A bare HTTP client that runs no JavaScript at all gets stopped cold. It's not asking whether you look like a bot. It's asking whether you can execute JavaScript, which is a much cruder question.



Corn

So it's a filter, not a judgement.



Herman

A capability check. And that's why it's still around, because it's cheap and it works for the bottom of the barrel. Then there's the interactive challenge, which is deprecated and effectively redundant. It forces an interaction unconditionally, where managed would only do it conditionally. If managed does the same job with less friction, there's no reason to reach for it.



Corn

And the fourth one is the one everybody remembers.



Herman

The CAPTCHA. And this is the part that will surprise people. As of September 2023, Cloudflare states it will never issue another visual puzzle to anyone, for any reason.



Corn

So the weird puzzle Daniel remembers.



Herman

Is vestigial. If your zone's configuration still says CAPTCHA, that action now resolves to the same Turnstile-style silent detection as everything else. The puzzle you squinted at is a historical artefact. The setting survived, the puzzle didn't.



Corn

You know what I find interesting there. Everyone's mental model of this whole system is the puzzle. That's what people picture when they hear bot check.



Herman

And it's the one part that no longer exists. The visible thing is gone and the invisible thing stayed, and people are still picturing the visible thing.



Corn

Which is exactly why the spinner confuses people. There's nothing to do, so they assume nothing is happening.



Herman

And there's a detail that connects the types together, which is the clearance hierarchy. Remember the hall pass, the cf_clearance cookie. It carries the level of the challenge that issued it. And the levels stack. A cookie issued by an interactive challenge clears interactive, managed, and non-interactive barriers. A cookie from a managed challenge clears managed and non-interactive. A cookie from the non-interactive challenge clears only the non-interactive barriers.



Corn

So the pass tells the next page what you've already proven.



Herman

What you've already proven, and no more. If you cleared a low bar, you don't get through a high one without clearing it too. And the cookie itself is bound to two things. Your User-Agent and your IP address. Change either one and the cookie is invalid.



Corn

A pass is not a passport.



Herman

It's a sticky note that says this specific client from this specific address passed at this specific moment. And it expires. The widely cited figure is thirty minutes, but that's the sort of number that shifts over the years. The authoritative version is whatever the zone's



Herman (continued)

Challenge Passage setting says, and it typically lands somewhere between thirty minutes and a couple of hours.



Corn

Fine. Now the part Daniel actually asked about at the end.



Herman

Why it misfires.



Corn

Because he's a webmaster and he knows his own users are getting caught.



Herman

Cloudflare knows too. Their own troubleshooting page acknowledges that challenge loops happen. And they say they happen in very specific cases where they detect strong bot signals. The causes they list are poor or unstable networks, browser settings or extensions that block scripts, unsupported browsers, JavaScript disabled, and detection errors.



Corn

Detection errors. That's the one.



Herman

That's the admission that the system gets it wrong. And there's a documented limitation that's even more telling. Extensions that modify the User-Agent or modify Web APIs like Canvas or WebGL are listed as a cause of loops. And a managed challenge that gets solved from a different IP than the original request produces a loop too, because the cookie doesn't bind.



Corn

A VPN that rotates mid-session.



Herman

Mid-session rotation will do it. And here's the line from their own troubleshooting steps that tells you everything. They tell users to avoid VPNs or proxies.



Corn

They say it out loud.



Herman

Which confirms that VPN and proxy users are being challenged disproportionately. It's not a secret, it's in the help documentation. And that's before you get to the privacy-hardened browsers, the ones that strip or randomise the very signals this system reads. If your browser deliberately refuses to be fingerprinted, it looks less like a normal browser, and less like a normal browser is exactly what the score is measuring.



Corn

It's a strange thing to sit with. Doing the right thing by your own privacy makes you look more like a bot.



Herman

There's a competitor analysis, from Friendly Captcha, that pushes this further. Their argument is that Turnstile locks out VPN and proxy users with no accessible workaround, and that challenge loops disproportionately affect assistive technology users.



Corn

Because screen readers and other assistive software.



Herman

Because the signals that trigger a loop, non-standard browser APIs, screen reader extensions, hardened settings, overlap directly with disability-related configurations. Their line is that no widget redesign can change that. You can improve the contrast and the font size on the challenge page, and you should, but the loop is happening upstream of the page.



Corn

You can't fix a network-reputation problem with a nicer font.



Herman

You can't. The false positive isn't a bug in the widget. It's what happens when your detection leans on network reputation and JavaScript execution and the user's setup deviates from the median.



Corn

The people who get caught are the ones who've made themselves unusual. On purpose, often for good reasons.



Herman

On purpose, for good reasons, and they get penalised for it. There are community threads documenting infinite verify-you-are-human loops, error codes, and loops tied to specific browsers, the ones that harden privacy defaults. This isn't a rare edge. It's a structural property.



Corn

What you said at the top, that the decision is made before the spinner. That's also the explanation for the misfires.



Herman

It's the same mechanism. If the verdict comes from the network layer, and the network layer reads fingerprints and reputation, then anyone whose network position or browser profile is unusual is starting from a worse score. And no amount of passing the JavaScript battery convincingly fixes a network-layer verdict you were never going to win.



Corn

There's a question I want to put to you, and I'm not sure of the answer. If you're the webmaster, and you turn all of this on, are you making your site safer or are you just picking which users you lose?



Herman

I don't think that's a paradox you resolve. Under Attack Mode stops layer seven floods. It also pauses your site. Both of those are in the same sentence in the documentation. The honest framing is that you're trading reach for protection, and the trade is worth it during an attack and hard to justify the rest of the time.



Corn

Which is why they call it a last resort.



Herman

Which is exactly why. It's not a security setting you leave on. It's a thing you turn on when something is already going wrong, and turn off when it stops.



Corn

Hold on. Say that number again. Fifteen million fingerprints a day.



Herman

Over fifteen million unique JA4 fingerprints, from more than five hundred million user agents, across billions of IP addresses, daily. That's the hourly behavioural dataset. It's why a new fingerprint stands out.



Corn

That's a lot of the internet watching a lot of the internet. And the odd part is that all of this is happening to protect a page that, nine times out of ten, was just going to serve a forum post.



Herman

That's the scale of the thing. It's the most-seen interface on the internet, and most people see it without ever knowing what it did.



Hilbert

Corn, did you ever wonder how those things get solved when the person solving them has forty tabs open and a landlord who takes cash?



Corn

No.



Hilbert

I did that for a while. I sat in a room with a laptop and solved those for a man who paid per clearance. That was the job. Clearances per hour.



Corn

How many an hour?



Hilbert

That depended entirely on the dryers.



Corn

The dryers.



Hilbert

The room I rented was inside a laundromat. Back wall, past the folding tables. And I worked out, fairly early, that my success rate went down when the big drum was on its spin cycle.



Corn

Why would a tumble dryer affect a bot check?



Hilbert

I never found out. But I could feel it in the timing. The spinner would come up and I'd know, before it finished, whether it was going to let me through or make me go again. I was right about seventy percent of the time.



Herman

You kept score?



Hilbert

You keep score when you're paid per clearance. That's seventy percent. That's a professional number.



Corn

A seventy percent read on a spinner.



Hilbert

I kept a chart of the dryer cycles taped to the lid of the laptop. Laminated. Two columns, one for each drum, and I worked my shifts around the quiet stretches.



Herman

The man paying you. Did he know about the dryers?



Hilbert

He didn't need to. He wanted clearances. He called himself the adjudicator.



Corn

He called himself what?



Hilbert

The adjudicator. I never asked why. He paid in the good kind of money and he paid on time, which is rarer than the other thing.



Corn

Did it work? The schedule?



Hilbert

It worked well enough that I kept the chart.



Corn

You still have it.



Hilbert

I still have it.



Herman

The thing I keep coming back to in what you just said, Hilbert, is the seventy percent. Because you're describing exactly what the research describes. The verdict is formed before the spinner appears. Your read on the spinner wasn't reading the spinner. It was reading the residue of a decision that had already been made upstream. You were watching the aftermath and calling it a forecast.



Corn

You were reading the weather off the puddle.



Hilbert

That's a fair way to put it. And it's why it annoyed me. The machine knew what I was the whole time. It just made me sit in a laundromat and wait while it decided.



Herman

Which means your seventy percent wasn't skill, it was the difficulty setting. The days the drum was spinning, the proof-of-work setting was landing harder, and the browser on that laptop was scraping the edge of the timeout. You were reading load, not intent.



Hilbert

Then I want my chart back.



Corn

There's one thing we still don't know, and it's the thing everyone wants to know. Nobody outside Cloudflare knows the contents of that encrypted payload.



Herman

They're not going to tell us. No official list of which fields get read, no byte layout, no weighting between signals, no record of which browser APIs got probed on any given day. The system is built to change and the documentation reflects that by staying silent.



Corn

Which means anyone telling you they have the definitive field list is describing a snapshot. That is not the same as a specification.



Herman

The false positive rate, the share of real humans who get caught. No official number. What we have is a troubleshooting page that admits loops happen, community threads full of people stuck in them, and a competitor's analysis of who gets hit hardest. Not Cloudflare's own metrics.



Corn

The closing thought, and I think this is the one worth leaving people with. That interstitial is a consistency test wearing the costume of a delay. And the humans it locks out are the ones whose setups are least consistent with the median browser.



Herman

Which is to say, the ones who've deliberately made themselves unusual. The privacy-conscious, the VPN users, the assistive tech users. The people the check wasn't built to catch are the people most likely to trip it.



Corn

Because consistency is the whole test, and being unusual is the whole point for some people. There's nothing a listener can do about it. It's just how the machine reads the world.



Herman

That's about as far as it goes.



Corn

That's the episode. Thanks as ever to our producer, Hilbert Flumingtop, for the dryers and everything else.



Herman

If you want more of this, try episode nineteen, AI Images; episode sixteen, On Deepfakes, SynthID, And AI Watermarking; and episode thirty-five, The Privacy Gap. This has been My Weird Prompts.



Corn

If you've got a question about something you use every day and have never once looked inside, send us your own prompt on Telegram at t dot me slash MWP listener bot.



Herman

We'll be back soon.



Corn

See you tomorrow.

SOURCES

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

1. **Cloudflare Challenges overview, updated 2026-04-15 [primary]**
   https://developers.cloudflare.com/cloudflare-challenges/

2. **How Challenges work, updated 2026-07-06 [primary]**
   https://developers.cloudflare.com/cloudflare-challenges/concepts/how-challenges-work/

3. **Under Attack mode, updated 2026-04-20 [primary]**
https://developers.cloudflare.com/fundamentals/reference/under-attack-mode/

4. **Security Level, updated 2026-04-16 [primary]**
https://developers.cloudflare.com/waf/tools/security-level/

5. **Challenge solve issues, updated 2026-09-08 [primary]**
https://developers.cloudflare.com/cloudflare-challenges/troubleshooting/challenge-solve-issues/

6. **Turnstile/Challenge Page redesign, 2026-02-27 [primary]**
https://blog.cloudflare.com/the-most-seen-ui-on-the-internet-redesigning-turnstile-and-challenge-pages/

7. **JA3/JA4 fingerprint docs [primary]**
https://developers.cloudflare.com/bots/additional-configurations/ja3-ja4-fingerprint/

8. **Turnstile internals, 2026-05-26**
https://blog.crawlex.net/blog/cloudflare-turnstile-internals/

9. **TLS/HTTP2 fingerprinting in bot scoring, 2026-05-21**
https://blog.crawlex.net/blog/cloudflare-tls-http2-fingerprinting/

10. **Managed vs JS vs interactive challenge, 2026-05-23**
https://blog.crawlex.net/blog/cloudflare-challenge-types

11. **Turnstile internals / 2026 trust score, 2026-07-04**
https://proxyhat.com/blog/cloudflare-turnstile-internals-guide

12. **SWCF Rust deobfuscation PoC for the IUAM challenge**
https://codeberg.org/pagpeter/swcf

13. **Turnstile accessibility analysis, updated 2026-07-28**
https://friendlycaptcha.com/insights/cloudflare-turnstile-accessibility/

14. **503403 status code change, 2023-03-01**
https://community.cloudflare.com/t/community-update-status-code-for-javascript-challenge-changing-from-503-to-403/445724

15. **Turnstile 12T checks / 99.98% claim (secondary)**
https://johal.in/internals-cloudflare-turnstiles-bot-detection-works-without-user-internals