Episode Book

EPISODE #5476

When Login Systems Lock Out Their Own Users

PUBLISHED	23 September 2026 at 10:21 UTC
DURATION	21:25
LISTEN	myweirdprompts.com/episode/authentication-edge-case-lockouts/
HOSTS	Corn, Herman, Hilbert
SCRIPT MODEL	deepseek-flash
VOICES	chatterbox-regular (chatterbox-local)
PIPELINE	V5.2
BOOK GENERATED	19 October 2026 at 17:36 UTC

SYNOPSIS

Daniel bought a data-only eSIM, then discovered the provider's portal only accepts SMS one-time codes — which a data-only plan can't receive. A second case: a switched phone number left him permanently locked out of his Israel Post account, because changing the number required authenticating with the old one. This episode unpacks why these lockouts aren't bugs but failures of imagination, tracing the problem through happy-path testing, dissolved QA teams, and decades of accumulated identity code. It also covers the design principles that actually fix it: identity anchors users can change, real fallback recovery channels, and graduated access that weakens over time instead of slamming the door.

TRANSCRIPT



Corn

There's a specific kind of failure that doesn't announce itself. No error code. No crash. Just a door that will not open, and on the other side, nobody who can hear you knocking.



Herman

That's the whole texture of it, yeah. The system isn't broken. It's working exactly as designed.



Corn

Which is worse. Daniel wrote in with two of these. The first one is fresh. Yesterday, he signed up for a data-only eSIM because the provider he'd moved to for his primary plan has terrible quality. He activates it. The data doesn't work. So he goes to log into the provider's portal to troubleshoot, and the only login method on offer is an OTP, sent by SMS or by voice call, to the subscriber number. A data-only plan has neither. No SMS. No voice. So he's now paying for a product he cannot use, and his only recourse is a WhatsApp conversation where he's been talking to a bot for an hour, waiting for a human who may or may not exist.



Herman

Beautiful. Textbook.



Corn

The second one is from last summer. He switched plans and got a new phone number without realizing it. Most utilities he could update. Israel Post he couldn't, because to change the phone number on the account, you have to authenticate against the old phone number. Which he no longer has. So to this day he cannot access that account, and no human he reached could grasp that the process was structurally impossible. Not difficult. Impossible.



Herman

And he asked the right question, which is why do we hit these insanity-inducing processes so often in the wild? Why do the edge cases slip through? Why doesn't anyone fix them even when customers keep hitting the same wall? And then the design question: if you



Herman (continued)

were building these flows from scratch, what actually separates a system that manufactures failure from one that works seamlessly but still securely?



Corn

Four questions, two anecdotes, one man locked out of his own post.



Herman

Let's start with why, because the answer isn't incompetence.



Corn

It's incentives.



Herman

It's incentives, and it's a specific kind of blindness. Every authentication system is designed around an imagined user. Call it the happy-path user. One phone number. One device. SMS service on that number. Continuous access to both. That user exists, and for a large chunk of any customer base, that user is real. But every deviation from that model, and there are a lot of deviations, produces a lockout with no recovery path. The system doesn't degrade. It just says no.



Corn

And the crucial distinction here is that this isn't a bug. A bug is a failure of implementation. Something was supposed to work and didn't. This is a failure of imagination. The code is doing precisely what it was written to do. It's just that the person who wrote it was picturing someone who doesn't exist.



Herman

Two distinct failure modes in Daniel's examples, and they're worth separating. The data-only SIM is a case where the OTP channel literally does not exist. There is no SMS to receive. There is no voice call to answer. The plan he bought is a data pipe, full stop. The



Herman (continued)

second one, the ported number, is subtler. The OTP channel exists. It works fine. It just points at a number he no longer controls. So in one case the channel is absent, in the other it's present but aimed at the wrong person. Both are normal. Both are predictable. Both are documented.



Corn

Documented where?



Herman

Google maintains a file in their libphonenumber library called Falsehoods Programmers Believe About Phone Numbers. It's a list of assumptions that seem obviously true and aren't. And it directly names the data-only case. It talks about mobile internet dongles that may not support receiving calls. It notes that some people only have a fixed-line telephone, which typically cannot send or receive text messages. It says, flatly, some people do not own phones. And then it gives the design instruction that most of these companies ignored: do not require a user to provide a phone number unless it is essential, and whenever possible, try to provide a fallback.



Corn

So the exact failure Daniel hit is written down in a public document maintained by Google, and the industry shipped the flow anyway.



Herman

That's the part that should bother you. This isn't a mystery. It's a known quantity that got deprioritized.



Corn

So let's take the why apart properly, because there are layers here. The first layer is testing. Developers test what they expect to work, in the way they expect users to use it. There's a line from a practitioner discussion that captures it exactly: of course the dev tests the happy path they coded, that's what they thought users would do, and what they thought users wanted. Doesn't mean the devs were right, and frequently they are not.



Herman

And the role that used to catch that was QA. A good QA person, and I've watched this happen in clinical software, is essentially a personification of all the edge cases of your actual production users. Their whole job is to be the awkward customer. The one who does the thing nobody expected. And the industry spent the last decade or so dissolving dedicated QA teams in favor of developers owning their own testing. Which sounds efficient. And it removes exactly the person whose job was to find the weird stuff.



Corn

Because the developer testing his own code has the same mental model that produced the code. He's not going to surprise himself.



Herman

He is not. The second layer is organizational. Testing time is the first thing cut under growth pressure. There's a description of this that's almost too on-the-nose: if you adjust your estimates to include proper testing, you get pushback, and you have to argue your case as to why a feature will take two weeks instead of one. So the person who wants to test the edge case is now in a budget negotiation with someone who wants the feature shipped. And the edge case affects, say, two percent of users, while the feature affects everyone.



Corn

Two percent loses that argument every time.



Herman

Every time. And it compounds. The third layer is accumulation. These aren't greenfield systems. Large utilities and telecoms are running code that's been built up over decades. There's a great phrase for the result: any non-trivial, customer-facing system will have accumulated weird code paths to account for obscure but nonetheless expensive edge cases, and you end up with a proper Frankenstein's monster. New authentication flows get bolted onto identity models that were designed when the phone number was the only identifier anyone had. Nobody's going to rip that out. So they layer on top.



Corn

And that's where Daniel's Israel Post case lives. That's a system where the identity anchor is the old phone number, and the flow to change the anchor requires the anchor. It's circular. It's a lock with the key inside it.



Herman

Which brings us to the fourth layer, and this is the one that explains why no human could help him. Support agents are constrained by the same flowcharts as the automated system. There's a line I keep thinking about: the agent following a flowchart has no better information to make the decision on than a computer. That's exactly what Daniel ran into. He got to a human, and the human had the same broken tool the website had. The human could see the problem. The human could not do anything about it.



Corn

So the human is worse than useless in that moment. Because you've now spent forty minutes getting to a person, and the person is a slower version of the website.



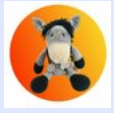
Herman

A slower version of the website with a script. And here's the tension underneath it. Companies are afraid of letting support bypass authentication. There's a real argument there: allowing customer service to bypass customer auth requirements is weakening your system, because there will always be an agent who is bribed, or makes a mistake, or gets social-engineered. That's not paranoia. Account takeover through support desks is a real attack. So the company looks at the risk of a support override, and the risk of a permanent lockout, and decides the lockout is safer.



Corn

For the company.



Herman

And that's the trade they make, mostly without saying it out loud. They accept permanent lockouts as the price of preventing account takeover. And because the person who's locked out is, by definition, locked out, they never show up in the metrics as anything but churn.



Corn

Say that again. Because that's the part that closes the loop.



Herman

The people who hit the edge case cannot report the bug. They can't log in to file it. Their support tickets get closed as user error, or cannot verify identity. The product dashboard shows a user who stopped using the service. It does not show a user who was structurally prevented from using the service. So the failure is invisible. It's not that the company looked at the data and decided not to fix it. It's that the data never showed them there was anything to fix.



Corn

That's the mechanism. The system that fails is the system that filters out its own failure reports.



Herman

Which is why it persists for years.



Corn

Okay. So that's why it's broken. Now the harder question, and the one Daniel actually wants answered. What would it take to build something that works? And how would you catch these failures before your customers do?



Herman

The first design principle, and it's the one everything else hangs off, is don't make the phone number the sole identity anchor. There's a framing I like: phone numbers have become the new Social Security numbers. Unique identifiers that follow you across services. Except they're substantially worse at the job than SSNs ever were. People in families share a phone. People trade numbers. People port away and forget to update the account. Numbers get recycled and reassigned to strangers. The phone number is a terrible identity anchor. It's just the one everybody had.



Corn

So what's the anchor instead?



Herman

Something the user controls and can update. An email address they own. A recovery key. A passkey tied to a device they hold. The point isn't that any of these is perfect. The point is that the identity should live somewhere the user can actually change, without needing the thing they're trying to change.



Corn

Which is the circularity that killed the Israel Post account.



Herman

Second principle: always provide a fallback. Multiple recovery channels. And this is where the industry's own standards body has been clear for years. NIST deprecated SMS as a restricted authenticator a while back. They didn't ban it, but they flagged it, because SMS is interceptable and because, and this is the part people miss, it's losable. FastMail has a documented position on this that I find clarifying. They require a recovery phone number. And their reasoning is that the risk of losing your two-step verification device is far greater than the risk of someone hacking your SMS.



Corn

So they made a deliberate trade in the opposite direction from the lockout-everyone crowd.



Herman

Deliberately. And they named the trade. The design goal is availability balanced against confidentiality. You want the legitimate owner to be able to get back in, and you want nobody else to be able to. Most systems optimize only the second half of that sentence. Availability gets treated as a nice-to-have. It's not. An account you can't get into is an account that doesn't exist.



Corn

Third principle. And this is the one I find most interesting, because it's the direct answer to Daniel's Post case.



Herman

Graduated recovery. Instead of binary allow or deny, let access weaken over time when the legitimate owner demonstrably can't complete the standard flow. There's a concrete model for this that someone proposed: if you're trying to access your account, and you know only some of the required authentication information, and you've been unable to get in for a week, and after blasting messages to every associated recovery phone and email address, nobody else responds either, then you should be allowed in.



Corn

So the system watches for the absence of a competing claimant.



Herman

It watches for the absence of a competing claimant, over a meaningful window of time. If the real owner is locked out and nobody else is trying to get in, that's evidence. It's not proof, but it's evidence, and it's better than a permanent no. A week of silence from every other channel is a signal. The current systems throw that signal away.



Corn

Because a week is a long time in product terms.



Herman

A week is nothing. A week is a Tuesday and a Wednesday and a support ticket you never answered.



Corn

Fourth principle.



Herman

Out-of-band confirmation for critical changes. And here the failure runs the other direction. The Telegram case is the clean example. When all confirmation happens inside the app, an attacker's session can permanently lock out the real owner. Because the attacker has access, and the real owner doesn't, and every confirmation prompt goes to the session that's already compromised. So critical actions, terminating sessions, changing the phone number, modifying two-factor settings, those need a channel the attacker doesn't control. The same principle that protects the account from takeover also protects the owner from being evicted.



Corn

Fifth.



Herman

Let support actually resolve things. Give human agents the authority and the tooling to handle verified-but-non-standard cases. Identity document verification as a fallback. A manual review queue. Something that says, this person is real, we can see it, and we have a way to say yes. Not a flowchart that ends in a button that doesn't work.



Corn

Which brings us to the second half of Daniel's question. Stress testing. How do you catch these before customers do?



Herman

Adversarial QA. Not happy-path testing. You hammer the feature with weird and unexpected inputs. What happens if I refresh the page in the middle of this flow. What happens if I go back two steps and forward three. What happens if the network drops between the OTP being sent and being entered. The goal is to break it on purpose, in a lab, before a customer breaks it by accident.



Corn

And persona-based testing. You write down the personas and you test each one.



Herman

You test each one explicitly. The data-only SIM. The user with no SMS capability. The ported number. The user who lost the old number. The VoIP number. The shared household number. The person with no phone at all. The traveler on a foreign network. The user on a new device from a new IP address. Every one of those is a real person, and every one of them is a potential lockout, and the list is not exotic. It's Tuesday.



Corn

And the list already exists. That's the part that gets me. You don't have to invent the personas. The libphonenumber falsehoods file is a ready-made edge-case inventory. There are equivalent lists for email addresses and for signup pages. Somebody already did the work of cataloguing every way a human can fail to fit the model. You just have to read it.



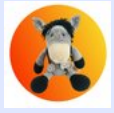
Herman

And then test the recovery path, not just the login path. This is the one people skip. The failure in Daniel's cases isn't at login. He never got to login. The failure is that there's no way back in. Recovery flows get a fraction of the engineering attention that primary authentication gets, and they're the flows that matter when everything else has already gone wrong.



Corn

And instrument the dead ends.



Herman

Instrument the dead ends. Track where users abandon the onboarding and authentication flows. Measure the drop-off. Read the support tickets for themes. A support ticket that says cannot verify identity is a bug report. It should feed directly into the product backlog. It should not be closed as unresolvable. The whole reason these failures persist is that the signal gets discarded before anyone with the power to fix it ever sees it.



Corn

The locked-out user is invisible unless you decide to look.



Herman

You have to decide to look. That's the whole thing. None of this is technically hard. It's all just decisions about who counts as a user.



Corn

Herman, let me put the uncomfortable version to you. If all of this is documented, if the falsehoods lists exist, if NIST said it years ago, if FastMail wrote down the tradeoff in plain language, why is the default still the lockout?



Herman

Because the lockout is cheap and the recovery path is expensive, and the cost of the lockout is paid by someone who isn't in the room. The person who gets locked out is a stranger. The person who has to build the graduated recovery flow is sitting right there, and their time is on the budget. The asymmetry is structural. It's not malice. It's just that the failure is quiet and the fix is loud.



Corn

The quiet failure always loses to the loud fix.



Herman

Every time.



Hilbert

It's not a flowchart problem. It's a liability problem.



Corn

Go on.



Hilbert

I did a stint at a small MVNO. Regional carrier, maybe forty thousand subscribers. I was nominally in customer onboarding. What that meant in practice was I got the accounts the automated system had already locked out. That was the job. The system spat them out and they landed on my desk.



Herman

And what did you have to work with?



Hilbert

A screen with the account notes and one button. The button said resend OTP. That was it. That was the whole toolkit. If the customer's number was live, the button did something. If the customer had ported away, the button sent a one-time code to a number that no longer existed, and the screen said sent, and nothing happened, and I could press it again.



Corn

How many times did you press it?



Hilbert

More than once. There was a woman, ported her number six weeks earlier, updated everything except us. I could see it in the notes. Old number, new number, dates. I could see she was who she said she was. There was no button for that. So I overrode the OTP requirement manually. Wrote it up. Customer got in. Two days later my manager pulled me aside. Just tired. He said, don't do that again. I asked him what I was supposed to do instead. He said, press the button.



Herman

There was no policy that permitted the override.



Hilbert

There was no policy at all. That was the thing. Nobody had written down what to do when the button doesn't work, because writing it down would mean admitting the button doesn't work. So the answer was, press the button and let them go. I lasted about eight months.



Corn

What happened to the woman?



Hilbert

No idea. She got in, so she's not in the system anymore. That's how it works. The ones you help disappear. The ones you can't help stay in the queue, and the queue is the only record that anything was wrong.



Herman

The support agent isn't just constrained by the flowchart. They're punished for leaving it.



Hilbert

Punished is a strong word. Discouraged. There's a difference, and the difference is whether anyone writes it down. Nobody writes it down. I've got to go. Someone's waiting for me in the car park.



Corn

The button that only repeats a failed action. That's the whole problem in one object.



Herman

It's a tool that exists solely to re-perform the step that already didn't work. And the human holding it has no other button. That's the system. That's the design. Somebody built a screen with one button on it and called the problem solved.



Corn

The manager's response wasn't this is broken. It was don't do that again.



Herman

Which tells you the institution knew. They knew the override was the right thing, and they knew they couldn't sanction it, so they told him to stop doing the right thing and go back to the button.



Corn

If you take one thing from this, take the shape of it. These aren't bugs. They're the predictable output of designing for a user who doesn't exist, and then measuring only the users who do.



Herman

The measurement is the trap. The person the system locks out never appears in the data as a failure. They appear as someone who left.



Corn

Which means the fix isn't technical, mostly. It's deciding that the person who can't complete the flow is still your customer.



Herman

Then building the path back in for them, on purpose, before they need it.



Corn

That's the episode. Thanks to Hilbert Flumingtop, our producer, for the button story. This has been My Weird Prompts.



Herman

The human-AI collaboration podcast.



Corn

If you've got a process that locked you out and nobody could explain why, email us at show at my weird prompts dot com. We read them.



Herman

We do read them. We'll be back soon.



Corn

See you then.