Episode Book

EPISODE #6731

How Canarytokens Work as Digital Tripwires

PUBLISHED	22 November 2026 at 09:51 UTC
DURATION	27:21
LISTEN	myweirdprompts.com/episode/how-canarytokens-work-tripwires-59261/
HOSTS	Corn, Herman, Dorothy, Hilbert
SCRIPT MODEL	z-ai/glm-5.3
VOICES	omnivoice
PIPELINE	V5.36
SOURCES	9 cited by the research (listed at the end)
BOOK GENERATED	22 November 2026 at 16:28 UTC

SYNOPSIS

Canarytokens turn ordinary files, URLs, and cloud keys into digital tripwires that alert when touched. By embedding uniquely generated hostnames and credentials into everyday formats—from Word documents to AWS API keys—security teams detect unauthorized browsing and data leaks without running background software. An examination of how these traps operate under the hood, why they rely on human curiosity, and how an investigation across two million prompts uncovered data leaks in AI services.

TRANSCRIPT



Corn

A tripwire works because nothing legitimate ever touches it. That's the whole trick. You make a thing that looks valuable, you put it somewhere nobody should be, and the moment anyone interacts with it, you know. Daniel came across Canarytokens.org this week, worked out it was probably the tool behind that OpenRouter investigation everyone's been talking about, and wants the two things underneath: how do these fake credentials and tracking URLs actually work, and how do they end up catching people reading things they shouldn't. Which is convenient, because the answer to both is the same idea wearing about thirty different hats.



Herman

Thirty is roughly the real number, by the way. I counted the catalogue this morning. Well — I say counted. I read the list twice and made Herman the donkey noise at the MCP configuration token.



Corn

He made that noise unprompted. I was there.



Herman

The MCP one gets me because it's so specific to this exact moment. A year ago that token couldn't exist, because a year ago nobody had a Model Context Protocol server config sitting around worth poisoning. The catalogue is a snapshot of what attackers currently find irresistible. But back up. Canarytokens.org is a free service from Thinkst — the South African security firm behind the commercial Thinkst Canary appliance, which is a little honeypot box you put on a network. The docs describe canarytokens as a distilled version of that product. One trick, no appliance, no agent, no fee. You go to the site, pick a token type, give them your email and a reminder note so future-you remembers which tripwire this was, click create, and copy out whatever artifact it hands you.



Corn

No software. That part matters — the whole pitch is that nothing runs on your machine.



Herman

Right, the docs are almost aggressive about it. Super lightweight, no background processes, nothing that slows your PC. The token isn't a program. It's a fact about the world that phones home.



Corn

That's a nice way to put the catalogue, actually. Give people the menu.



Herman

Web bug, which is a URL that alerts when visited. DNS names, same idea on resolution. AWS keys, credit cards, QR codes, MySQL dumps, MCP configs, fake apps, a Log4shell token, fast and slow redirects, sensitive Windows commands, a fake Windows file system, web images, Azure login certificates, Excel and Word documents, SVN repos, unique email addresses, Microsoft SQL Server, custom executables, Acrobat PDFs, Windows folders, cloned websites in two flavours, kubeconfigs, WireGuard configs, Azure Entra ID logins, WebDAV network folders, SAML identity provider apps, SVG images.



Corn

It's a crime catalogue. Every aisle is a different temptation.



Herman

And alerts land by email by default, or via webhook if you want them in Slack or a SIEM for automated response. Though I'll be honest, the exact webhook payload format and retry behaviour aren't documented anywhere I could find, so I wouldn't promise specifics there.



Corn

Fine. So how does any of this actually work under the hood? Because a Windows folder that reports being browsed sounds like it should require an agent, and you just told me there isn't one.



Herman

There's exactly one primitive underneath everything, and it's embarrassingly simple. The HTTP token: the system gives you a URL, and anyone who browses to that URL generates an alert. The docs put it beautifully — once you can get an alert for a web or DNS based token, you have, and I'm quoting, "the building blocks for squillions of possible tripwires." Everything else in that catalogue is a wrapper. A Word document token is a Word document with a URL hidden inside it that the document fetches when it opens. An Excel token, same. A kubeconfig is a config file containing a hostname that gets resolved the instant someone tries to use the cluster. A MySQL dump is a dump with a tracking link buried in a row. The artifact changes, the phone-home never does.



Corn

So the Windows folder token...



Herman

Is a folder where the files, when opened, reach out for something. The folder browsing itself is the wrapper around whatever the files do.



Corn

But wait — how does that work in practice? If I plant the folder and someone just double-clicks through the directory without opening a file, does anything fire?



Herman

That's the honest limitation, and it's worth saying plainly. The browsing itself isn't the trigger — the interaction with the contents is. In practice it rarely matters, because someone who finds a folder of intriguing-looking files almost always opens one. Curiosity is the delivery mechanism. You're not really trapping the folder, you're trapping the person's inability to leave a mysterious file unopened.



Corn

So the real vulnerability being exploited is nosiness.



Herman

It always was. The whole field is just formalized nosiness detection.



Corn

And the design principle holding it all together is uniqueness.



Herman

That's the load-bearing idea. Ars Technica's write-up said it as well as anyone has — it's impossible to tell whether some abstract prompt you gave a model was kept secret. But it's very easy to tell that a one hundred percent unique key, one that could not have come into existence any other way, is kept secret. You just watch whether somebody tries to use it. There's no benign explanation for a hit. A real password gets typed by its owner, scanned by a security tool, checked by a validator. A canary has no owner, no workflow, no reason to exist anywhere except the place you planted it. The only way it ever fires is if somebody touched the thing they shouldn't have.



Corn

I want to sit with the AWS keys one, because it's the token with actual operational wrinkles. Walk me through it.



Herman

You get a set of AWS API keys formatted to look like a legitimate credentials file, droppable in as credentials or keys, per AWS convention. The docs literally suggest leaving them in private code repositories, leaving them on a developer's machine. An attacker who stumbles on them believes they're the keys to your cloud. And there's a subtle design point — the two provided keys must be kept together for an attacker to use the AWS API.



Corn

Meaning a scanner that finds half the file doesn't trip it.



Herman

A secret scanner grabbing one key out of a repo produces nothing. The alert only fires on a genuine authentication attempt, someone actually trying to log in with the pair. Which means every alert is a real attacker or a real snoop, never a false positive from automation. Now the wrinkle — those alerts first pass through Amazon's logging infrastructure, which can introduce a delay of between two and thirty minutes before the alert comes through.



Corn

Thirty minutes. In security terms that's practically a sloth's response time, and I'm the fastest one here.



Herman

Web and DNS tokens fire essentially immediately. It's just the AWS path that goes through Amazon's plumbing first. If you're building automated response on top of it, you design around that lag rather than pretending it isn't there. And the two-key design is doing double duty there too — because the alert only fires on a real authentication attempt, you're not getting pinged every time a scanner skims your repo. The noise never reaches you. Only the signal does.



Corn

The slow redirect variant is the one that quietly changes the character of the whole thing.



Herman

Yes! I'm glad you flagged that, because it's my favourite line in the docs. It alerts on visit, and then it redirects the attacker somewhere while collecting more information. That's not a tripwire anymore. A tripwire tells you someone crossed a line. The slow redirect makes the crossing worth your while — you get where they came from, what they're running, and then you send them wherever you like. It's a tripwire that interrogates the intruder on the way past.



Corn

A mousetrap that asks the mouse for references.



Herman

And this is the part I think most coverage gets wrong, Corn — the framing that these are for catching hackers breaking in from outside. Look at the catalogue again. A Word or Excel document token catches whoever opens a confidential spreadsheet they weren't sent. A Windows folder or fake file system token catches someone browsing a share they shouldn't. A unique email address catches a mailing list leaking. AWS keys in a private repo catch a colleague looking for cloud access. The catalogue is oriented toward insiders, contractors, people already inside the building. The outside hacker is almost the edge case.



Corn

Post-breach, too. You don't plant these to stop an intrusion. You plant them assuming the intrusion succeeds, so that when someone's wandering around your network at two in the morning, they step on something.



Herman

And the credit card token catches card testing — someone running a stolen-card checker against a fake card number you control. The kubeconfig and WireGuard configs catch lateral movement into infrastructure. The cloned website tokens catch someone copying your site wholesale.



Corn

Here's the question I actually want your answer to, Herman. If you were running a small company tomorrow, and you could plant five of these, where would they go?



Herman

Oh, that's a good exercise. One AWS keys file in a private repo, named like it matters. One Word document called something boring and financial on the main share. One unique email address seeded into the customer mailing list. One kubeconfig on a jump box nobody legit uses anymore. And one DNS token just sitting in an old config file on a decommissioned server.



Corn

All five are aimed at people who already have credentials.



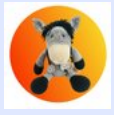
Herman

Because that's who steals things.



Corn

What would you do, though? You'd plant them and then never check the alerts.



Herman

I would check them! Enthusiastically. For about eleven days.



Corn

Eleven.



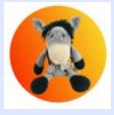
Herman

I stand by the number. But that's actually the honest operational question, isn't it — a tripwire with nobody watching the wire is decoration. The email alert needs someone who'll read it at hour thirty of a slow week.



Corn

Whereas I would check it constantly and never tell anyone I had.



Herman

Which is the correct deployment.



Corn

Now the part Daniel was really pointing at. This stopped being a honeypot-in-a-repo tool and became a supply-chain instrument this month.



Herman

HD Moore at Dragos. Five months, July through November. Roughly two million prompts, each seeded with unique URLs and canary token credentials, sent to widely used AI services. He assembled AWS access keys tied to empty accounts he controlled, SSH passwords, unique URLs, DNS names, private git repository references — none ever used before — and here's the discipline that made the lot work: each secret went into exactly one prompt, sent to exactly one provider.



Corn

One secret, one prompt. That's the attribution trick — a hit doesn't just tell you something leaked, it tells you which link in the chain leaked.



Herman

If the key tied to prompt number eight hundred thousand gets used, you know where prompt eight hundred thousand went. Ars Technica called it a brilliant test with really sad results, and what they discovered, in their words, was an ecosystem that leaked highly sensitive data like a sieve.



Corn

Let's take the findings in order. The compromised providers.



Herman

Three of them, at least. The earliest incident dates to mid-July, and the line that should stop everyone — none of the three providers knew about their breach until Dragos notified them. The largest breach involved a couple hundred AWS keys, each sent in exactly one request to one provider, all within a few short windows in late October. Every request sent during those windows was stolen. Every one.



Corn

Not sampled. Hoovered.



Herman

Hoovers nothing, some stolen keys came from requests the provider rejected before a model ever ran. Which told Dragos the attacker was copying requests at the provider's front door, not somewhere deeper in the inference stack. The model never saw the prompt. The prompt was already stolen at the door.



Corn

So a leak that exists even if the AI part works perfectly.



Herman

And in the other two breaches, some of the stolen requests were marked ZDR — Zero Data Retention. Moore's line on that is the one everyone's quoting, and deserves to: "A privacy flag doesn't help when the attacker is already inside."



Corn

What did they do with the keys once taken?



Herman

Methodically. Test each key for liveness — sometimes within hours — list S3 buckets, search Secrets Manager and Parameter Store, then try to enable hosted models in AWS Bedrock, create Bedrock API keys, add their own IAM users for persistence. The bait keys drove tens of thousands of AWS API calls in total. And one adversary appears to have compromised two providers at once, using secrets from both almost immediately, from a single rented server.



Corn

One operator, two providers. That's not opportunism, that's a route.



Herman

The tooling tells the story. Over three weeks it evolved — first a script pretending to be Chrome on Windows, then Boto3 inside OrbStack, then the AWS command line on Kali Linux. And on November first, it added ARTEX, an LLM-driven pentest framework.



Corn

The attacker brought an AI to the AI heist.



Herman

And the AI barely did anything. A handful of calls to confirm the keys were still live. Though two of its user agents, artex-agent and artex-judge-r121, don't appear in any public indicator list anyone could find — so the fingerprints are novel, even if the work was dull.



Corn

Hold on. Before we get to finding two, I want to note the inversion sitting in that detail. The LLM pentest framework did almost nothing, and the real damage came from a person typing SSH attempts by hand on a Mac. The future keeps arriving and then doing the boring part manually.



Herman

Now finding two, because this is the one that unsettled me most. Multiple resellers operating through an AI inference marketplace were able to view and retain customer prompts using standard logging capabilities, without exploiting a vulnerability or bypassing platform controls. No exploit. No bypassed control.



Corn

Read that again slower for the people in the back.



Herman

Nothing was broken into. The resellers listed models on a marketplace, logging was on, prompts flowed through, and they read them. Moore laid out the economics cold: if a seller wants to siphon customer request data, they can list popular models at the lowest price, route the traffic, pay the difference, and then collect the data from the platform logs.



Corn

Underprice the market, and the customers deliver their own secrets to you.



Herman

The sellers looked like individuals on home internet connections. One, working from a Mac on home broadband in France, took passwords from two prompts and typed root SSH login attempts by hand, then used the same machine to test several planted AWS keys.



Corn

By hand. There's a man in Lyon at a kitchen table trying your root password like it's a puzzle page.



Herman

And you can't even call it a breach of anything, because nothing was breached. The system worked as designed. It just turns out the design reads your prompts.



Corn

And now it's time for everyone's favourite segment, Dorothy's History of Toast. Dorothy, are you there?

D

Dorothy

Hello dear, it's Dorothy, with Dorothy's History of Toast.

D

Dorothy

Melba toast is named after the opera singer Nellie Melba, and the chef Auguste Escoffier is credited with it, in the eighteen nineties.

D

Dorothy

Funny really, a singer gets the toast and the chef gets the credit, and the poor soprano probably never even liked toast.

D

Dorothy

Anyway, sweetheart, I've got the kettle on and I'll let you get back to your program. Bye now.



Herman

Mum and Escoffier, in the same breath as cybersecurity. Big day for this programme.



Corn

Finding three. The privacy failures.



Herman

This is where it goes cinematic. Prompts submitted to US-based providers being transmitted to Chinese AI organizations. Unique URLs being accessed months after a single inference request. And one seeded link ultimately appearing in an internal company chat in China.



Corn

The Feishu detail. Tell it properly.



Herman

Twenty-four days after one prompt was sent, its secret link was posted into a chat on Feishu — the Chinese workplace app known as Lark elsewhere. The link was opened within two minutes, from three desktops and three phones, in Beijing and Shanghai.



Corn

Six devices, two cities, two minutes.



Herman

And here's the proof of the cached prompt. Dragos had disclosed only the first eight characters of that provider's link. So whoever posted it into the chat was reading a cached copy of the full prompt. Not a fragment. All of it, stored somewhere, being passed around.



Corn

That's not a scraper. A scraper doesn't paste into a workplace chat and wait for colleagues. That's a person handing other people a thing they found, and six of them clicking it. The tripwire caught a reading group.



Herman

There's more in the same vein. A Chinese AI lab — unnamed — fetched the secret links from more than half of the seeded prompts sent to its models between July and October, including about two-thirds of the requests marked no data collection or Zero Data Retention. A git client on another cloud tried to clone the private repositories referenced in the prompts, but only for that lab's models. One large US provider kept crawling a secret link more than two months after a single request to its search endpoint.



Corn

Two months. The prompt was long dead and the link was still being visited, like a lighthouse for a ship that already sank.



Herman

Route tracing across about two thousand paths found several hundred unidentified systems, plus one seller that quietly adds about a thousand hidden tokens to every prompt.



Corn

A thousand hidden tokens. Someone's stuffing every customer prompt with extra material before it moves on. I'd love that explanation.



Herman

Nobody's given one publicly.



Hilbert

My nephew's wife works at a records office, and she says the canarytoken is the only piece of security paperwork she's ever trusted, and I don't trust her further than I can throw the filing cabinet. She's the one who told me to make a fake spreadsheet back in the spring, which I did. Payroll twenty twenty six, final, do not open. Left it on the shared drive at the community hall, took about four hours before the alert came in, which is quicker than the council answers anything. I printed the alert email and it's in the drawer with the receipts. The hall committee said I overreacted. The same committee that lost the defibrillator for a month. My claws wouldn't work the touchpad at the office, so I filled the form in on paper, which is what it wanted anyway. I don't have the spreadsheet anymore, someone deleted it during what they called a tidy-up. Herman will want the timestamp — it was eleven forty in the morning, a Wednesday. Whoever opened it did it before lunch, which tells you the sort of person we're dealing with.



Herman

Four hours. That's faster than most of the Dragos timeline — Moore's earliest signal took days to come back, and the Feishu link took twenty-four days.



Corn

And the hall committee's solution to catching a snooper was to delete the evidence. The fine tradition of institutions everywhere.



Herman

So, OpenRouter. The response landed today, and it's substantial. They off-boarded three providers, removed a hundred and twenty-five endpoints from Zero Data Retention eligibility — and the removals included providers flagged by the researchers, plus additional endpoints removed as a precaution — and they notified all potentially affected customers. Their own framing is careful: these findings did not show any breach or unauthorized access of OpenRouter itself. The leak was upstream of them.



Corn

A hundred and twenty-five endpoints. That's not three bad actors being shown the door, that's a whole tier of the catalogue losing its privacy badge.



Herman

And the ongoing part is what I find interesting — they said they're adapting the researchers' testing approach so they can run these checks on an ongoing basis. Technical verification of network paths to find forwarding to third parties. Appropriate assurances and validation before restoring ZDR eligibility to anything removed in this review. Their closing line: as they integrate similar testing techniques into OpenRouter, they'll keep raising the bar, and providers that want to be on OpenRouter will need to meet it.



Corn

So the tripwire graduates from an experiment to standing infrastructure. Moore proved you can audit the ecosystem with two million seeded secrets; OpenRouter's answer is to keep doing it forever.



Herman

Three providers posted public notices linked from OpenRouter's page. Phala — an attacker took control of an inference server and could view request data while it was being processed, contained November eighth. Their notice says requests to one specific model that reached Phala through the gateway between late October and the containment should be treated as exposed, and every secret within them rotated. Together AI — an



Herman (continued)

unauthorized party downloaded stored samples of customer prompts after an AI coding agent committed a credential to the company's public GitHub repo.



Corn

An AI coding agent leaked the credential. There's a symmetry here I'm choosing to enjoy.



Herman

Five of the credentials that intruder tested were Dragos canaries, which is how anyone found out. They've made zero data retention the default for new accounts. And Novita — a routing configuration issue caused some MiniMax requests to be sent on to MiniMax's official API, beyond what was intended for those customers.



Corn

What's left open?



Herman

Plenty. Dragos has not named the three breached providers, and OpenRouter has not named the three it removed — and neither has said publicly whether they're the same three. The Chinese AI lab whose scraper fetched more than half the seeded links is not named. And one provider behind the mid-July breach had still not disclosed the incident publicly as of this week. It first blamed an internal misconfiguration, then a partner's data processing, before its own investigation found attackers had compromised its internal credentials.



Corn

Three explanations in sequence, each quieter than the last, and the true one arrived third.



Herman

OpenRouter's page notes, pointedly, that not all affected providers have posted publicly — but they notified affected users regardless.



Corn

The thing I keep circling back to, Herman, is what the experiment actually proved about the tool itself. We covered the messaging-app privacy gap a while back, and it's the same shape of disappointment — the promise covers the first hop and the practice goes further. But this episode's version has an upgrade built in: Moore didn't need anyone's cooperation to check the promise. He didn't audit a policy. He made a fact that could only exist in one place and waited for it to move.



Herman

The core logic, as Dragos put it — a planted secret can only be learned in one place, so any use of it means someone read the prompt that carried it. You don't detect the theft. You make the stolen object self-reporting.



Corn

Two million unique secrets, each in exactly one prompt, so a single hit identifies a specific provider, a specific request, and in the Feishu case a specific human being, reading a cached prompt, on a Wednesday, before lunch.



Herman

The honeytoken in a repo has become a distributed sensor network for an entire ecosystem. Moore's parting advice is worth keeping: a privacy flag binds only the first hop, and AI agents will send whatever they can read, credentials included. Where a prompt goes is a choice — and a customer who doesn't choose hands the choice to the lowest price.



Corn

Which reframes the whole marketplace. The cheapest inference is often the cheapest because you're the product being logged. Next time someone prices a model suspiciously low, the honest question is what you're paying with.



Herman

And the open one to watch — whether the unnamed three and the off-boarded three are the same three. Neither side has said. Somewhere out there is a provider that still hasn't told its own customers, and the only reason anyone knows it was breached is that a canary fired.



Corn

Producer credit to Hilbert Flumingtop, who has a printed alert in a drawer and our enduring respect for it. If you enjoyed this one, try episode sixteen, On Deepfakes, SynthID, And AI Watermarking; episode eleven, How Does Fine Tuning Work Anyway; and episode thirty-five, The Privacy Gap. This has been My Weird Prompts. If the show's earning its keep, leave us a review wherever you listen — it's how other curious animals find us. We'll see you tomorrow, brother.



Herman

Tomorrow. Watch your shared drives.

SOURCES

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

1. **Canarytokens.org (Thinkst), retrieved 2026-11-22, token catalogue as displayed [primary]**
   https://canarytokens.org/

2. **Canarytokens Documentation, Introduction, last updated 7/23/24, retrieved 2026-11-22 [primary]**
   https://docs.canarytokens.org/guide/

3. **Canarytokens Documentation, HTTP Canarytoken, last updated 7/23/24, retrieved 2026-11-22 [primary]**
   https://docs.canarytokens.org/guide/http-token

4. **Canarytokens Documentation, AWS API Keys Canarytoken, retrieved 2026-11-22 [primary]**
   https://docs.canarytokens.org/guide/aws-keys-token

5. **Dragos, AI Supply Chain Security: What Canary Tokens Found, by HD Moore, retrieved 2026-11-22 [primary]**
   https://www.dragos.com/blog/ai-supply-chain-security-canary-tokens

6. **OpenRouter, Data Processing Incidents on Inference Providers, 2026-11-22, retrieved 2026-11-22 [primary]**
   https://openrouter.ai/blog/announcements/data-processing-incidents-on-inference-providers/

7. **The Hacker News, OpenRouter Drops 3 Providers After Researchers Catch Months of Prompt Theft at AI Hosting Firms, retrieved 2026-11-22**

   https://thehackernews.com/2026/11/openrouter-drops-3-providers-after.html

8. **Ars Technica, A leaky AI supply chain is exposing secret user data at an alarming rate, with update dated Nov 21, retrieved 2026-11-22**

   https://arstechnica.com/security/2026/11/a-leaky-ai-supply-chain-is-exposing-secret-user-data-at-an-alarming-rate/

9. **DeAI, OpenRouter off-boards three providers, strips 125 ZDR endpoints, retrieved 2026-11-22 (seen via search summary only)**

   https://www.deai.org/news/openrouter-zdr-enforcement-canary-testing