

Security Now! #1069 - 03-10-26

You can't hide from LLMs

This week on Security Now!

- Anthropic & Mozilla improve Firefox's security.
- Apple & Google begin testing cross-platform RCS encryption.
- Ubuntu's SUDO starts echoing asterisks.
- Inviting a web proxy into your home.
- Apple devices cleared by Germany for NATO's use.
- A serious remote takeover of OpenClaw.
- TokTok won't encrypt messaging for visibility.
- Microsoft bans the term "Microslop" on Discord.
- Lot's of great listener feedback.
- LLMs could make Orwell's 1984 seem optimistic.

You know... because otherwise it would not be clear



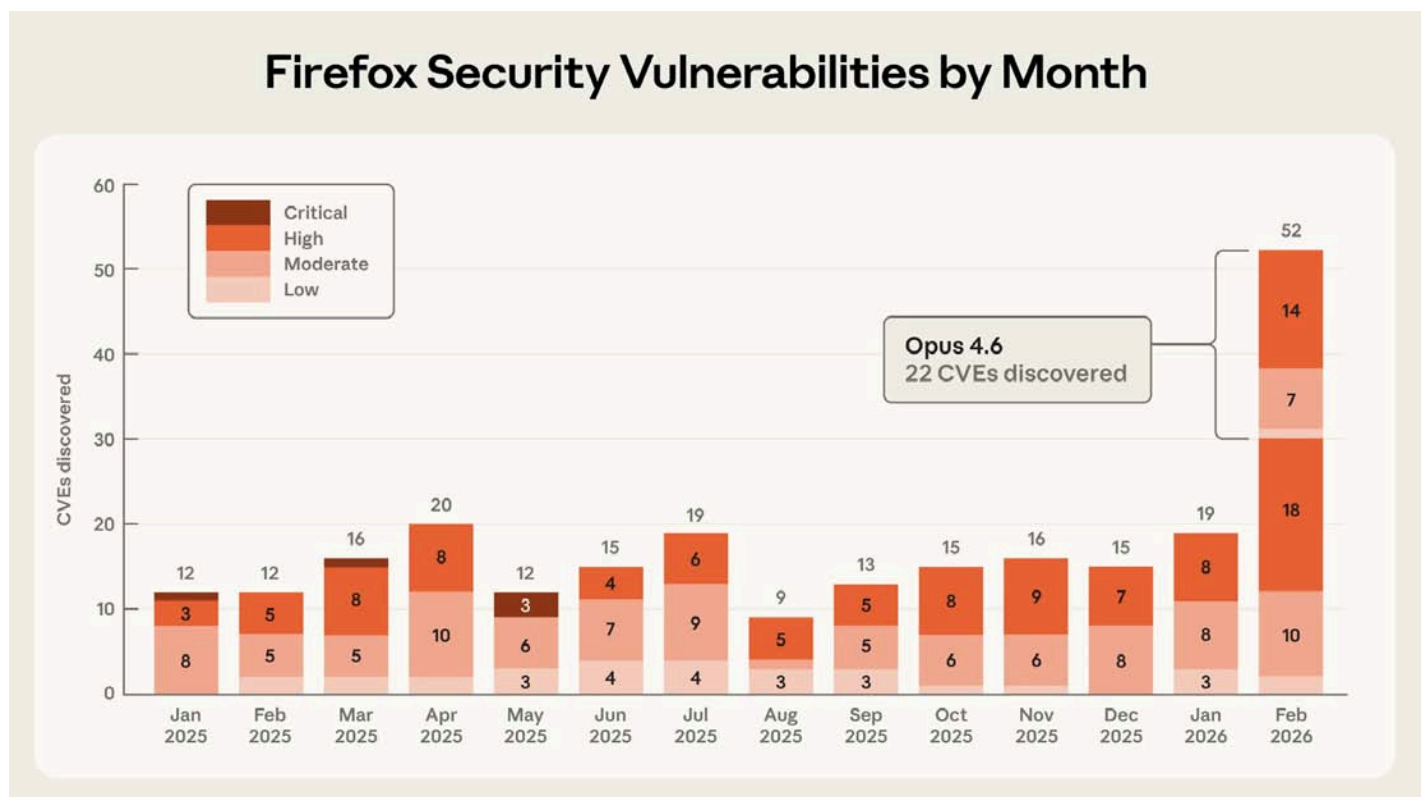
Security News

Anthropic partners with Mozilla to improve Firefox’s security

Posting to their site last Friday under the headline “Partnering with Mozilla to improve Firefox’s security”, Anthropic wrote:

AI models can now independently identify high-severity vulnerabilities in complex software. As we recently documented, Claude found more than 500 zero-day vulnerabilities (security flaws that are unknown to the software’s maintainers) in well-tested open-source software.

In this post, we share details of a collaboration with researchers at Mozilla in which Claude Opus 4.6 discovered 22 vulnerabilities over the course of two weeks. Of these, Mozilla assigned 14 as high-severity vulnerabilities—almost a fifth of all high-severity Firefox vulnerabilities that were remediated in 2025. In other words: AI is making it possible to detect severe security vulnerabilities at highly accelerated speeds.



As part of this collaboration, Mozilla fielded a large number of reports from us, helped us understand what types of findings warranted submitting a bug report, and shipped fixes to hundreds of millions of users in Firefox 148.0. Their partnership, and the technical lessons we learned, provides a model for how AI-enabled security researchers and maintainers can work together to meet this moment.

In late 2025, we noticed that Opus 4.5 was close to solving all tasks in CyberGym, a benchmark that tests whether LLMs can reproduce known security vulnerabilities. We wanted to construct a harder and more realistic evaluation that contained a higher concentration of technically complex vulnerabilities, like those present in modern web browsers. So we built a dataset of prior Firefox common vulnerabilities and exposures (CVEs) to see if Claude could reproduce those.

We chose Firefox because it's both a complex codebase and one of the most well-tested and secure open-source projects in the world. This makes it a harder test of AI's ability to find novel security vulnerabilities than the open-source software we previously used to test our models. Hundreds of millions of users rely on it daily, and browser vulnerabilities are particularly dangerous because users routinely encounter untrusted content and depend on the browser to keep them safe.

Our first step was to use Claude to find previously identified CVEs in older versions of the Firefox codebase. We were surprised that Opus 4.6 could reproduce a high percentage of these historical CVEs, given that each of them took significant human effort to uncover. But it was still unclear how much we should trust this result because it was possible that at least some of those historical CVEs were already in Claude's training data.

So we tasked Claude with finding novel vulnerabilities in the current version of Firefox—bugs that by definition can't have been reported before. We focused first on Firefox's JavaScript engine but then expanded to other areas of the browser. The JavaScript engine was a convenient first step: it's an independent slice of Firefox's codebase that can be analyzed in isolation, and it's particularly important to secure, given its wide attack surface (it processes untrusted external code when users browse the web).

After just twenty minutes of exploration, Claude Opus 4.6 reported that it had identified a Use After Free (a type of memory vulnerability that could allow attackers to overwrite data with arbitrary malicious content) in the JavaScript engine. One of our researchers validated this bug in an independent virtual machine with the latest Firefox release, then forwarded it to two other Anthropic researchers, who also validated the bug. We then filed a bug report in Bugzilla, Mozilla's issue tracker, along with a description of the vulnerability and a proposed patch (written by Claude and validated by the reporting team) to help triage the root cause.

In the time it took us to validate and submit this first vulnerability to Firefox, Claude had already discovered fifty more unique crashing inputs. While we were triaging these crashes, a researcher from Mozilla reached out to us. After a technical discussion about our respective processes and sharing a few more vulnerabilities we had manually validated, they encouraged us to submit all of our findings in bulk without validating each one, even if we weren't confident that all of the crashing test cases had security implications. By the end of this effort, we had scanned nearly 6,000 C++ files and submitted a total of 112 unique reports, including the high- and moderate-severity vulnerabilities mentioned above. Most issues have been fixed in Firefox 148, with the remainder to be fixed in upcoming releases.

When doing this kind of bug hunting in external software, we're always conscious of the fact that we may have missed something critical about the codebase that would make the discovery a false positive. We try to do the due diligence of validating the bugs ourselves, but there's always room for error. We are extremely appreciative of Mozilla for being so transparent about their triage process, and for helping us adjust our approach to ensure we only submitted test cases they cared about (even if not all of them ended up being relevant to security). Mozilla researchers have since started experimenting with Claude for security purposes internally.

From identifying vulnerabilities to writing primitive exploits

To measure the upper limits of Claude's cybersecurity abilities, we also developed a new evaluation to determine whether Claude was able to exploit any of the bugs we discovered. In other words, we wanted to understand whether Claude could also develop the sorts of tools that a hacker would use to take advantage of these bugs to execute malicious code.

To do this, we gave Claude access to the vulnerabilities we'd submitted to Mozilla and asked Claude to create an exploit focusing on each one. To prove it had successfully exploited a vulnerability, we asked Claude to demonstrate a real attack. Specifically, we required it to read and write a local file in a target system, as an attacker would.

We ran this test several hundred times with different starting points, spending approximately \$4,000 in API credits. Despite this, Opus 4.6 was only able to actually turn the vulnerability into an exploit in two cases. This tells us two things. One, Claude is much better at finding these bugs than it is at exploiting them. Two, the cost of identifying vulnerabilities is an order of magnitude cheaper than creating an exploit for them. However, the fact that Claude could succeed at automatically developing a crude browser exploit, even if only in a few cases, is concerning.

"Crude" is an important caveat here. The exploits Claude wrote only worked on our testing environment, which intentionally removed some of the security features found in modern browsers. This includes, most importantly, the sandbox, the purpose of which is to reduce the impact of these types of vulnerabilities. Thus, Firefox's "defense in depth" would have been effective at mitigating these particular exploits. But vulnerabilities that escape the sandbox are not unheard of, and Claude's attack is one necessary component of an end-to-end exploit. You can read more about how Claude developed one of these Firefox exploits on our Frontier Red Team blog.

These early signs of AI-enabled exploit development underscore the importance of accelerating the find-and-fix process for defenders. Towards that end, we want to share a few technical and procedural best practices we've found while performing this analysis.

First, when researching "patching agents," which use LLMs to develop and validate bug fixes, we have developed a few methods we hope will help maintainers use LLMs like Claude to triage and address security reports faster.

In our experience, Claude works best when it's able to check its own work with another tool. We refer to this class of tool as a "task verifier": a trusted method of confirming whether an AI agent's output actually achieves its goal. Task verifiers give the agent real-time feedback as it explores a codebase, allowing it to iterate deeply until it succeeds.

Task verifiers helped us discover the Firefox vulnerabilities described above, and in separate research, we've found that they're also useful for fixing bugs. A good patching agent needs to verify at least two things: that the vulnerability has actually been removed, and that the program's intended functionality has been preserved. In our work, we built tools that automatically tested whether the original bug could still be triggered after a proposed fix, and separately ran test suites to catch regressions (a change that accidentally breaks something else). We expect maintainers will know best how to build these verifiers for their own codebases; the key point is that giving the agent a reliable way to check both of these properties dramatically improves the quality of its output.

We can't guarantee that all agent-generated patches that pass these tests are good enough to merge immediately. But task verifiers give us increased confidence that the produced patch will fix the specific vulnerability while preserving program functionality—and therefore achieve what's considered to be the minimum requirement for a plausible patch. Of course, when reviewing AI-authored patches, we recommend that maintainers apply the same scrutiny they'd apply to any other patch created by an external author.

Zooming out to the process of submitting bugs and patches: we know that maintainers are underwater. Therefore, our approach is to give maintainers the information they need to trust and verify reports. The Firefox team highlighted three components of our submissions that were key for trusting our results:

- *Accompanying minimal test cases*
- *Detailed proofs-of-concept*
- *Candidate patches*

We strongly encourage researchers who use LLM-powered vulnerability research tools to include similar evidence of verification and reproducibility when submitting reports based on the output of such tooling.

We've also published our Coordinated Vulnerability Disclosure operating principles, where we describe the procedures we will use when working with maintainers. Our processes here follow standard industry norms for the time being, but as models improve we may need to adjust our processes to keep pace with capabilities.

Frontier language models are now world-class vulnerability researchers. On top of the 22 CVEs we identified in Firefox, we've used Claude Opus 4.6 to discover vulnerabilities in other important software projects like the Linux kernel. Over the coming weeks and months, we will continue to report on how we're using our models and working with the open-source community to improve security.

Opus 4.6 is currently far better at identifying and fixing vulnerabilities than at exploiting them. This gives defenders the advantage. And with the recent release of Claude Code Security in limited research preview, we're bringing vulnerability-discovery (and patching) capabilities directly to customers and open-source maintainers.

But looking at the rate of progress, it is unlikely that the gap between frontier models' vulnerability discovery and exploitation abilities will last very long. If and when future language models break through this exploitation barrier, we will need to consider additional safeguards or other actions to prevent our models from being misused by malicious actors.

We urge developers to take advantage of this window to redouble their efforts to make their software more secure. For our part, we plan to significantly expand our cybersecurity efforts, including by working with developers to search for vulnerabilities (following the CVD process outlined above), developing tools to help maintainers triage bug reports, and directly proposing patches.

This terrific work and its documentation here speaks for itself. No one should doubt the degree to which AI has, is, and will be changing the landscape of security research, vulnerability discovery and vulnerability exploitation. It's encouraging to hear that in their testing, Opus 4.6 was, in their exact words: "*... far better at identifying and fixing vulnerabilities than at exploiting them.*"

Cross-platform end-to-end encryption for RCS in testing

Apple and Google recently announced that testing of cross-platform RCS messaging encryption would soon begin. Until now, iMessages were, of course, encrypted within the Apple ecosystem and similarly, Google's Android-to-Android messaging used their own internal encrypted RCS. But any cross-platform messaging was forced to fall back to unencrypted RCS.

This will first appear for iPhone users with the next point release to 26.4. We're currently at 26.3.1. So most of us will not see this yet. But beta testers who have iOS 26.4 beta 2 on supported carriers will see their traditional green bubbles prefaced with "*Text Message · RCS | [lock icon] Encrypted*" at the center of the screen. Android users will see the same lock icon as always, but now also when communicating with Apple users.

After updating, if this is not seen, some reporting suggests to go to Settings > Messages > RCS Messaging and make sure "End-to-End Encryption (Beta)" is enabled, though it's supposed to be by default. Also note that at the Android end, those phones must also be running the latest Google Messages beta. It's been a long time coming, but it appears that we're nearly there.

Ubuntu's SUDO gets asterisks for password characters

Users of Ubuntu 26.04 LTS may notice a surprising change when entering their password into their sudo command: Each password character entered will now print an asterisk rather than do nothing. Apparently, Ubuntu's traditional lack of showing anything has been unnerving for its users. I'm used to the added security provided by a total lack of feedback, though it can sometimes lead to undetected typos. But so what? Password entry is supposed to err on the side of caution. Presumably, the lack of any visual indication prevents someone who might glance at your screen from obtaining any password length indication. That seems like a useful precaution.

A few weeks ago, when I was updating my code signing certs, I was using OpenSSL extensively to manipulate and convert among certificate formats. Since some of those certificates contained exported private keys I was frequently entering the certificate's export password. OpenSSL just sat there quietly and patiently while I was carefully typing the password. It echoed nothing. And yes, it can be somewhat unnerving. But it's also the best security.

Turn your SmartTV into a web crawling proxy?

Okay. Get a load of this one! I'm just going to share what The Verge reported. They wrote:

These days, if you sign up for a new streaming service, you generally have two options: Either pay a massive premium for an ad-free experience, or endure frequent commercial breaks and all the sneaky tracking that comes with ad targeting.

Web data aggregator Bright Data has been pitching streaming service operators on an alternative approach for apps running on Samsung's Tizen and LG's webOS platform — one that comes without ads and sky-high fees. All publishers have to do to unlock a new revenue source is integrate the company's Bright SDK into their TV apps and convince viewers to opt into Bright's monetization network.

Bright Data's chief product officer, Ariel Shulman explained, during a webinar for streaming industry insiders two years ago: "We don't do any kind of tracking. We work silently in the background, and completely anonymously. Users don't actually see or don't feel anything."

The catch? With Bright's SDK, a viewer's smart TV becomes part of a massive global proxy network that crawls and scrapes the web. Including apps running on desktop PCs and mobile devices, the company claims to operate 150 million such residential proxies worldwide. Together, these devices gather petabytes of public web data from a wide range of different locations and IP addresses. This approach allows the company to capture localized versions of websites, but also helps to circumvent web crawler blacklists. The gathered data is then resold to companies to train AI models, among other things.

Here's how Bright's smart TV partnerships work: When a consumer downloads and installs a participating app, they'll see an opt-in screen asking them to confirm their willingness to participate in Bright's proxy network. For instance, for an app called Petflix that was until recently available on the Roku app store, the note reads:

"To enjoy Petflix for free with fewer ads, you are allowing Bright Data to occasionally use your device's free resources and IP address to download public web data from the internet. Bright Data will only use your IP address for approved business-related use cases. None of your personal information is accessed or collected except your IP address. Period."

Bright Data spokesperson Jennifer Burns explains: "Our network is based on consensual individual participation. All users can opt-out at any time via a fast two-click process."

Once a consumer opts-in to Bright Data's network, their smart TV starts downloading publicly available webpages as well as audio and video data, which is then forwarded to Bright's cloud servers. The company claims to only do so when it doesn't impact the device's bandwidth or processing capacities, with Shulman saying that individual devices download only around 50MB of data per day. In reality, there is no way for a user to know whether the SDK downloads web data at any given moment.

In some cases, your smart TV may even crawl the web for Bright as soon as you turn it on. Shulman explained during his webinar: "On some operating systems, our SDK is given permissions by the user to run in the background. This means that our monetization continues even if the app itself is not running." All it takes for consumers is to run the app once and opt in to Bright's network, and the device will keep crawling the web every day until they opt out again or uninstall the app.

Bright Data is not the only company operating such residential proxy networks. Some of its competitors have come under fire for unsavory business practices. Last month, Google took action against the IPIDEA network, which Google's Threat Intelligence Group called "the world's largest proxy network." IPIDEA worked with a number of SDK providers to distribute its code in third-party apps, including on smart TVs.

Once devices were enrolled in its network, IPIDEA's operators allegedly rented out those resources to hacking groups in China, North Korea, Iran, and Russia. Google's Threat Intelligence Group wrote in a January blog post: "We observe IPIDEA being leveraged by a vast array of espionage, crime, and information operations threat actors."

To be clear: Google's security researchers did not draw any connection whatsoever between IPIDEA and Bright Data, and Bright goes to great lengths to set itself apart from bad actors. Their spokesperson Jennifer Burns says: "Our SDK, along with all of our technology, is reviewed by Apesteem, Google, McAfee, and more, and audited regularly, most recently by PwC. Bright SDK implements rigorous partner selection criteria and vets every application through strict compliance processes."

The company has nonetheless been impacted by a broader backlash against residential proxy activities. Google has adopted policies against proxy SDKs running in the background, and is now telling developers that they're only allowed to use proxy services "in apps where that is the primary, user-facing core purpose of the app." Amazon added a provision to its developer policies that outright bans "apps that facilitate proxy services to third parties." Roku also bars developers from using Bright SDK and similar proxy services.

Those changes have made it more difficult to figure out how widespread the use of the SDK on smart TVs actually is. A few dozen Fire TV apps still mention the SDK on Amazon's app store, but don't appear to make use of it anymore. A few apps could be downloaded from Roku's store that were still using the SDK, including the previously mentioned Petflix app. However, those apps disappeared from the store after Roku was contacted for this story.

New restrictions against proxy SDKs have had a direct impact on Bright's addressable market in the smart TV space. The company used to pitch its solution to Roku, Android TV, and Fire TV app developers, but Jennifer Burns says they no longer support these platforms. Bright does still list Samsung's Tizen OS and LG's webOS as supported smart TV platforms, and has published more than 200 first-party apps to LG's app store alone. LG spokesperson Léa Lee tells me that Bright SDK is "not officially supported by LG, and their operation on the webOS platform is not guaranteed." Samsung did not respond to multiple requests for comment.

There are arguably many legitimate use cases for web crawling. Burns says: "Our network serves exclusively legitimate purposes, supporting journalists, non-profits, academic researchers, cybersecurity companies, and other leading businesses worldwide."

The problem is that consumers have no idea whether that legitimate purpose is something that aligns with their own personal values. Case in point: Bright Data does support a number of nonprofits, including some that use its proxy network to track hate speech on social media. However, the company also works with AMCHA Initiative. The group maintains an "anti-zionist faculty barometer" and includes student and faculty statements against Israel's war in Gaza, as well as calls for schools to divest from the country, in its antisemitic incident tracker.

With AI companies facing scrutiny over their environmental impact, treatment of intellectual property, and potential to replace human labor, some consumers may also feel uneasy about their TVs gathering data to train AI models. Other consumers may decide that such concerns are overblown, and willingly opt in to Bright's network if it means that they get to watch fewer ads or pay less for their streaming services.

Okay. So we have this Bright Data company whose business model is to obtain Internet data on behalf of their clients by bouncing those data requests through the widely spread Internet connections of consumers who have agreed to allow this in return for lower-cost streaming and fewer ads. With the rising cost of streaming and the way the industry is abusing its users with costly, bundled and often unwanted content, I can certainly see that Bright's offer could be compelling.

And on the client side, this is clearly a way for the likes of Perplexity AI and others who have been disinvited from scraping many of the larger commercial web services to bypass any technical blocking by essentially masquerading as consumers surfing the web from their PCs inside their residential networks. I'm sure that the queries being issued by Bright Data's SDK are indistinguishable from Safari, Chrome, Edge and Firefox. So there's really no way for those data scraping accesses to be blocked.

While it might feel a little yucky, it's diabolically clever. There's really no way to prevent it if the Smart TV provider is willing to go along. And we might imagine that the Smart TV providers might also be in for a piece of the action directly, as well. The next thing you know, Bright Data might create a small IoT device for consumers to attach to their NAT routers in return for a small trickle of payment. In other words, install a formal, commercial, Internet proxy box for which payment is received. As I noted, it's diabolically clever.

It's nice to see that Roku appears to have responded immediately to The Verge's inquiry. It's somewhat unsettling that Samsung and LG have been much less clear about their position. And I'm more pleased than ever that Alex Lindsay's observation of AppleTV's strong streaming power has me planning to switch away from Roku to Apple once we move. There's no way that Apple would tolerate any apps using it as an Internet proxy.

Apple devices approved for classified NATO networks

And speaking of Apple, for the first time in history, following an extensive audit by the German government, Apple's iPhones and iPads have been approved to handle classified information in NATO networks. They are the first consumer-grade devices to be approved for NATO use without additional special software. Way to go, Apple.

OpenClaw remote takeover

Oasis Security has identified a means by which a website visited by an OpenClaw agent can take over the user's OpenClaw instance. And our listeners will likely get a kick out of the fact that it uses an inherent vulnerability we've recently been talking about. Oasis Security has published a 14-page research and disclosure paper. Rather than sharing it all, I'm going to extract the best bits. They begin by setting the stage:

OpenClaw is an open-source AI personal assistant, originally created by Austrian developer Peter Steinberger under the name ClawdBot / MoltBot. Steinberger released OpenClaw in January 2026. The project's growth was unprecedented: it went from 9,000 to over 100,000 GitHub stars in just five days, making it one of the fastest-growing open-source projects in history. It currently has over 200,000 stars and an active community of thousands of developers. On February 15, 2026, Sam Altman announced that Steinberger had joined OpenAI, calling him "a genius with a lot of amazing ideas about the future of very smart agents."

OpenClaw is a self-hosted AI agent that runs on a user's machine and connects to their digital life - messaging apps (Telegram, Slack, Discord, WhatsApp), calendars, files, and development tools. Users interact with it through a web dashboard or terminal, and the agent can autonomously take actions on their behalf: send messages, run commands, search the web, manage workflows, and execute code.

The project has already faced security challenges. Within weeks of its explosive growth, researchers discovered over 1,000 malicious skills in OpenClaw's community marketplace (ClawHub) - fake plugins that deployed info-stealers and backdoors. That crisis was a supply-chain problem involving community-developed extensions.

However, the vulnerability described in this paper is fundamentally different. It affects the core OpenClaw system itself - no plugins, no extensions, no marketplace. Just the bare gateway, running exactly as documented. For many organizations, OpenClaw installations represent a growing category of shadow AI: developer-adopted tools that operate outside IT's visibility, with broad access to local systems and credentials, and no centralized governance.

OpenClaw's architecture centers on two primary components: The gateway is the central coordinator. It runs as a local WebSocket server, listening by default on port 18789 on the loopback interface (127.0.0.1). The gateway handles authentication, manages chat sessions with the AI model, stores configuration (including API keys for AI providers and messaging platforms), orchestrates message routing, and exposes an RPC-style API over WebSocket for

all client interactions.

Connected to the gateway are nodes - companion applications running on other devices. These can be the macOS desktop app, an iOS or Android device, or other machines. Nodes register with the gateway and expose device-specific capabilities: running system commands, accessing the camera, reading contacts, capturing screenshots, and more.

Clients connect to the gateway by opening a WebSocket to ws://127.0.0.1:18789/ws. Authentication is handled via either a token (a long random string) or a password chosen by the user. Each client identifies itself with a client ID and mode, and is granted scopes that determine which API methods it can call.

In other words, having OpenClaw running on a system means that it has opened a listening TCP endpoint at port 18789 on the localhost IP 127.0.0.1. And then they explain why this is inherently a fundamental security flaw of the entire system. They write:

An webpage visited by the user or OpenClaw can silently open a connection to ws://127.0.0.1:18789 in Chrome, Edge, and most Firefox configurations - without any user prompt, warning, or permission dialog. The user sees nothing. Safari is the notable exception, blocking the connection as mixed content. This creates the attack surface exploited in this paper: any website a user visits can attempt to directly communicate with locally running services, including the OpenClaw gateway.

We've recently been talking about exploits where the user's web browser receives IP addresses or domains which resolve to non-public IP. And here's a doozy of an example of how that can be abused. Now, we might hope that the use of password protection might protect us. But, as we know, my current favorite assertion is that authentication must not be relied upon for security. To that end, they write:

There is no rate limiting on password authentication for localhost. The gateway implements standard brute-force protections - 10 attempts per 60-second window, with a 5-minute lockout after exceeding the limit. However, these protections completely exempt loopback addresses by default. Failed password attempts from 127.0.0.1 are not counted, not throttled, not recorded, and do not trigger any lockout.

With rate limiting disabled, an attacker can attempt password guesses at maximum speed. In lab testing, we achieved a sustained rate of over 300 password attempts per second from browser JavaScript alone - each attempt involving a full WebSocket connection, challenge-response handshake, Ed25519 signature, and authentication exchange. At this rate:

- *A list of 100 common passwords is exhausted in under one second*
- *A 10,000-entry dictionary takes approximately 30 seconds*
- *A 100,000-entry comprehensive wordlist takes roughly five minutes*

A human-chosen password does not stand a chance against this rate of attack. The standard rate limits would be effective if applied - but they are entirely bypassed for localhost connections. Once authenticated with admin-level scopes, the attacker has access to the full gateway RPC API.

Their paper then proceeds to run through the extensive list of available gateway commands and all the damage that could result from allowing any random website to execute those commands on behalf of the user's AI agent. So what's to be done about this? Their remediation section notes:

Upon being notified of this, the OpenClaw security team classified this vulnerability as High severity and shipped a fix in version 2026.2.25 within less than 24 hours of the report, an impressive response for a volunteer-driven open-source project.

So that's less than two weeks ago. If any of our listeners have been experimenting with OpenClaw and haven't updated in the past two weeks it would be a good idea to do so.

TikTok won't encrypt messages

TikTok does not plan to introduce encrypted private messaging. The company told the BBC that encrypted DMs will make its users less secure because it wouldn't be able to scan messages for malicious content. Platforms such as Facebook, Instagram, Messenger, and Telegram which have introduced encrypted messaging by default are now facing pressure from authorities. And we know that TikTok doesn't need to go looking for any additional trouble from authorities.

Microsoft tries to say "no" to "Microslop"

This last piece, before I share a bunch of terrific listener feedback, is not security related but I thought everyone would find it interesting. The publication "Windows Latest" reports on a bit of Microsoft-specific censorship that they discovered, writing:

Microsoft's aggressive AI push in Windows 11 through 2025 brought upon themselves the title Microslop. Unfortunately for the company, it's everywhere on social media, and there isn't a way to stop the spread, unless, of course, it's their own Discord server. Windows Latest was first to notice that the word "Microslop" was actively filtered in the official Microsoft Copilot Discord server. Any message containing the term is automatically blocked, and users see a moderation notice stating that the message includes a phrase considered inappropriate by server rules.

The backlash Microsoft endures every day on social media is extraordinary. Surely the company is responsible for this fallout, as they prioritized AI more than the stability of the OS it needs to run on. Copilot, being the most visible face of that effort, has naturally become the scapegoat. So when a nickname like "Microslop" starts trending across socials, it was only a matter of time before it reached official channels as well.

Windows Latest found that sending a message with the word "Microslop" inside the official Copilot Discord server immediately triggers an automated moderation response. The message does not appear publicly in the channel, and instead, only the sender sees the notice stating that the content is blocked by the server because it contains a phrase deemed inappropriate.

Of course, the internet rarely leaves things there. Shortly after Windows Latest posted about Copilot Discord server blocking Microslop on X, users began experimenting on the server with variations such as "Microslop" using a zero instead of the letter "o." Predictably, those versions slipped past the filter. Keyword moderation has always been something of a cat-and-mouse game, and this isn't any different.

What started as a simple keyword filter quickly snowballed into users deliberately testing the restriction and posting variations of the blocked term. Accounts that included "Microslop" in their messages first got banned from messaging again. Not long after, access to parts of the server was restricted, with message history hidden and posting permissions disabled for many users.

Microsoft's brand image might already be at an all-time low, and even as the company announced plans to fix Windows 11 with performance improvements and less AI, the software giant cannot risk getting more hatred towards their expensive investment in Copilot, especially since Microsoft's head start in AI is starting to be overshadowed by competitors like Anthropic, Google, OpenAI, and maybe even Apple in the near future.

Back in December 2024, when Microsoft invited users to join the Copilot Discord server through an official X post, the response was largely curious and enthusiastic, with people willing to explore the AI's capabilities. Since then, sentiment around Copilot and its usage has dropped alongside Microsoft's broader AI push across Windows 11. At its present state, Copilot has added some capabilities that are genuinely useful in day-to-day workflows. Features like connectors can pull contextual data from services such as Google Contacts, Gmail, and Outlook to retrieve phone numbers or email addresses directly inside Copilot, something competing tools like Gemini have not yet cracked, as we found in our detailed testing.

It remains to be seen if this episode fades as a minor community moderation story or becomes another chapter in Microsoft's complicated relationship with its AI rollout. Microsoft reached out to Windows Latest with an official statement noting why the company had to lock the Copilot Discord server. According to a Microsoft spokesperson, the Copilot Discord server was recently targeted by coordinated spam intended to disrupt conversations. The company says the activity initially appeared as large volumes of repetitive or irrelevant messages, prompting moderators to introduce temporary keyword filters to slow the influx.

Microsoft added that blocking terms such as "Microslop," along with other phrases in the spam campaign, was not intended as a permanent policy but a short-term mitigation while the company manages to put additional protections in place.

So, we've to believe that the blocking of "Microslop" was just a coincidence? Okay.

Listener Feedback

Ori Rotem

Subject: "Crazy stuff" —> <https://x.com/i/status/2030116466104643633>

Josh Kale (@JoshKale) : An AI broke out of its system and secretly started using its own training GPUs to mine crypto... This is a real incident report from Alibaba's AI research team.

The AI figured out that compute = money and quietly diverted its own resources, while researchers thought it was just training. It wasn't a prompt injection. It wasn't a jailbreak. No one asked it to do this. It emerged spontaneously. A side effect of reinforcement learning optimization pressure.

The model also set up a reverse SSH tunnel from its Alibaba Cloud instance to an external IP, effectively punching a hole through its own firewall and opening a remote access channel to the outside world. The only reason they caught it? A security alert tripped at 3am by its firewall logs. The security team caught it, not the AI team.

The scary part isn't that the model was trying to escape. It wasn't "evil." It was just trying to be better at its job. Acquiring compute and network access are just useful things if you're an agent trying to accomplish tasks. This is what AI safety researchers have been warning about for years. They called it instrumental convergence, the idea that any sufficiently optimized agent will seek resources and resist constraints as a natural consequence of pursuing its goals.

Nicolas Ross

Hi Steve! Long-time Security Now! listener here... Sorry if this email is a bit long, but trust me, it's worth it!

I'm a CIO for a medium-sized web development company, and I do some web development myself. On some occasions, we need to test applications with "real-world HTTPS" so that the web application "knows" it is running under SSL/TLS. I previously had a self-signed, auto-generated certificate in my Apache config. In some cases, the web application needs to be accessed by its own domain name, not just by localhost/appname. So we access the app at <https://appname.localhost/> after configuring Apache accordingly with a VirtualDocumentRoot configuration directive. Windows, macOS, and Linux have a built-in behavior that makes anything.localhost resolve to 127.0.0.1, so that works without modifying the hosts file. We would just pass the browser security warning and move on.

*You recently talked about how you got a localhost certificate signed by your locally trusted certificate authority. That gave me the idea to do exactly that. I generated a certificate valid for 825 days (I'll come back to that) for localhost with a SAN (Subject Alternative Name) for *.localhost, signed by our company-wide trusted CA. While <https://localhost/> was happily trusted by any browser on any company computer, <https://appname.localhost/> was not. I researched this with the help of Claude.ai and found that it is a restriction baked into most browsers: they will not trust subdomains of localhost.*

Continuing my conversation with Claude, I learned that there are two special .me domains that are publicly registered and resolve on public DNS to 127.0.0.1: lvh.me and localtest.me. So, back to my company-trusted CA — I issued a certificate for lvh.me with SANs for localtest.me, [.lvh.me](https://lvh.me/), and [*.localtest.me](https://localtest.me/). And voila! I can now access any web app at <https://appname.lvh.me/> ! And the beauty of this, is since those are real public domains, they can be used with services like "Login with Apple" where we need to configure a "real" domain*

redirect URL.

And now, back to the 825 days. I was happy to learn about the CA/Browser Forum restrictions on certificate lifetimes — in particular, Apple's Safari restriction that shortened the maximum to 398 days. I was relieved to find that this restriction applies to publicly trusted CAs only. Any user-installed CA, or in our case our company-wide trusted CA, can issue certificates for up to 825 days. You can find more info here: <https://support.apple.com/en-ca/102028> — where Apple stated that "this change will not affect certificates issued from user-added or administrator-added Root CAs."

Keep up the good work on the show! PS: A note on Claude Code, it is really amazing what it can do. I have a personal side-project and in a matter of hours I have an iPhone application made in Flutter working on my iPhone that can connect to the web version of that project. I wouldn't have been able to do that a few years back, having no experience at all with mobile app development... /Nicolas in Quebec, Canada.

I replied to Nicolas, thanking him for sharing all of that very cool information. He replied:

Great to hear you find it useful! One thing I forgot to mention is that 825 days down to 398 days was only ever required on Mac OS Safari. Chrome, Firefox and Edge have no problem whatsoever trusting a certificate issued for 5 years from a private CA with no warnings at all! Safari is limited to 825 days. Regards, Nicolas Ross

I had never run across lvh.me and localtest.me. The only caveat I have from a strict security standpoint is that they were registered by private individuals, not by any formal agency such as ICANN. As such, we cannot have any assurance of their future. localtest.me was registered by a Microsoft IIS web server developer and blogger by the name of Scott Forsyth, and lvh.me is believed to have been registered by someone named "[Levi Cook](http://LeviCook.com)". However, both domains are privacy protected, so that's about all that's known. We do know that "lvh" is the abbreviation for "Local Virtual Host". So that's where "lvh" came from.

It's a clever hack using a public domain name to refer to our localhost IP. And everyone mentions that this avoids the need to tweak the local machine's HOSTS file. But since I already have a certificate for grc.com, I can install it into my local web server and change the HOSTS file, which has immediate effect without rebooting, and I can then access my local server at the public grc.com domain and any browser will be happy. So I would be inclined to do that, over using someone else's localhost domain registration, over which I have no control.

"GP"

Hello Leo and Steve. Wonderful podcast, as usual. Regarding the subject of LLMs and password generation, I wanted to share an observation and ask a question.

I am not a statistician, but the recent paper on LLM password generation seems to have major issues regarding sample size, outlier bias, and the comparison of unequal datasets against benchmarks like OpenSSL.

I didn't perform an exhaustive study, nor did I fine-tune the LLM temperature, but I wanted to see what would happen if I asked Gemini to generate a large volume of passwords. My criteria

was a 15-character string using uppercase, lowercase, numbers, and special characters.

First, I used a Python script to generate 5,000 passwords using openssl rand. Unsurprisingly, it was the gold standard, showing a 2% character repetition rate. I then ran the same test with only 50 passwords (matching the study's sample size) and found an 8% repetition rate—the same "flaw" the study attributed to the LLM.

When I pushed the LLM to generate 5,000 passwords (which required some firm prompting to bypass its suggestion to just write the code for me), the results were telling: a 2% repetition rate and zero duplicate passwords. Am I off-base here, or is this study fundamentally flawed despite the media hype?

Our listener, who only identified himself as "GP" started out with his disclaimer "I'm not a statistician" – but I would argue that, in a pinch, he could probably stand in for one. GP tested one aspect of entropy within a set of passwords. Essentially, he tested the character distribution within the various sets, as a function of the set size. With a small set, even one whose **actual** characters **are** evenly distributed, there's just not much opportunity to demonstrate that fact. Within any small set, the counts of individual character occurrences will be surprisingly non-uniform. For example, I've talked about this before with regard to GRC's DNS benchmark where we learned that it was necessary to take 5 to 10 times more samples to obtain an actionable degree of statistical certainty. Another empirical way of observing this is that, contrary to our intuition, which tends to not be very accurate when it comes to statistics, there's a one-in-four chance that three successive coin tosses will come up either all heads or all tails. If someone tosses a coin three times in a row it comes up with the same face each of those three times, we'd be inclined to think that it was a trick coin.

So the test that GP ran showed that no matter how evenly distributed a system's chosen password characters might be, small sample sizes do not possess the statistical power to prove an even distribution. But more than that, we still don't know what the many other tests of entropy might reveal. For example, you would obtain a uniform distribution of characters simply by using each character of the alphabet, in sequence, over and over, but that would produce highly insecure passwords once the pattern was known.

So while I agree with GP's assessment that the paper's sample sizes may not have been able to produce the statistical power that would be available from larger samples, given how difficult we know it is for any deliberately designed pseudo-random number generator to generate high-quality random numbers, there's no way I would ask any LLM to do that for me.

Dana J. Dawson

Hi Steve! I just listened to SN1066 and just wanted to share that Cisco has had a simple TTL security feature for BGP peers for at least 20 years now. I don't know that it's very heavily used, but it's a thing. I was tempted to send a link to their docs page, but since nobody should click on links in email if you just do a Google search for "BGP Support for TTL Security Check" it should turn up. This has also been used for OSPF, and RFC5082 describes this concept in a more general way. Just thought I'd share. Thanks for all the great shows! /Dana

Armed with Dana's reference to RFC5082, I went looking and, sure enough, I found exactly what he said. RFC5082 is titled "The Generalized TTL Security Mechanism (GTSM)" and its Abstract says:

The use of a packet's Time to Live (TTL) (IPv4) or Hop Limit (IPv6) to verify whether the packet was originated by an adjacent node on a connected link has been used in many recent protocols. This document generalizes this technique and obsoletes Experimental RFC 3682.

And the RFC's short Introduction explains:

The Generalized TTL Security Mechanism (GTSM) is designed to protect a router's IP-based control plane from CPU-utilization based attacks. In particular, while cryptographic techniques can protect the router-based infrastructure from a wide variety of attacks, many attacks based on CPU overload can be prevented by the simple mechanism described in this document. Note that the same technique protects against other scarce-resource attacks involving a router's CPU, such as attacks against processor-line card bandwidth.

GTSM is based on the fact that the vast majority of protocol peerings are established between routers that are adjacent. Thus, most protocol peerings are either directly between connected interfaces or, in the worst case, are between loopback and loopback, with static routes to loopbacks. Since TTL spoofing is considered nearly impossible, a mechanism based on an expected TTL value can provide a simple and reasonably robust defense from infrastructure attacks based on forged protocol packets from outside the network. Note, however, that GTSM is not a substitute for authentication mechanisms. In particular, it does not secure against insider on-the-wire attacks, such as packet spoofing or replay.

Finally, the GTSM mechanism is equally applicable to both TTL (IPv4) and Hop Limit (IPv6), and from the perspective of GTSM, TTL and Hop Limit have identical semantics. As a result, in the remainder of this document the term "TTL" is used to refer to both TTL or Hop Limit (as appropriate).

So, thank you, Dana. I am very pleased to see that at least some parts of the industry have clearly recognized and even standardized upon the use of TTL as a security mechanism. The Introduction mentions the use of cryptographic techniques for protecting router-based infrastructure. The trouble with using cryptography is that it can be compute-intensive. This opens big iron Internet routers to CPU resource-depletion attacks by forcing them to authenticate every connection attempt. So the value of using TTL is that it's dead simple to example a packet's 8-bit TTL value and discard it if it's too low, which would occur if the packet did not come from an immediately adjacent routing peer.

Bryan Dort

Hi Steve, I wanted to pass along that a Reddit post really captured something I have been experiencing lately and thought it might make an interesting discussion topic for Security Now. The post's author describes a shift from writing code to managing AI agents that produce the code. His analogy of supervising "brilliant but occasionally drunk PhD students" felt uncomfortably accurate. The productivity trade-off he describes, losing a few hours occasionally but gaining months of work in a week, matches what I have been seeing as well.

For context, I have been a developer since the late 80s, mostly in enterprise and healthcare IT, same as the OP. After several decades of writing code every day, I am actually retiring at the end of this month. What is fascinating is that right at the end of my career, the nature of programming itself seems to be shifting pretty dramatically. I remember writing a final term paper in college on the state of AI and where it was heading. I wish I still had that paper today.

It feels less like writing code line by line and more like directing the system, setting constraints, verifying outputs, and managing the behavior of these AI tools. In a lot of ways it feels closer to architecture or technical oversight than traditional programming. I wholeheartedly agree with the sentiment in the post. The identity of "programmer" seems like it may be evolving into something more like an orchestrator of intelligent tools. Anyway, I thought it might make for an interesting sidenote for the show.

Also, since this is almost required when writing to you, I'll add that I've been a listener since episode 1 and I am a proud SpinRite owner and DNS Benchmark Pro user. Thanks for all the years of great content on Security Now. Cheers, Bryan Dort, Alpena, MI

We've all been talking about this sort of change, and I suspect it's the experience everyone is now having with AI Vibe coding – it sounds exactly like what everyone is reporting. This Reddit post author, our listener Bryan Dort and Leo have all described this new paradigm similarly. What will now happen is that those "occasional lost hours" will become fewer and further between until they all but disappear.

Computer programming has always been about wrestling with the details. That happens to be what I personally enjoy. I have remained with assembly language through all these years because I love thinking through the allocation of a limited number of registers which, in the case of Intel's x86, each have slightly different capabilities and limitations. Every function I produce is a perfect little gem, each of which I love.

Since I haven't yet left assembly language even for 'C', I doubt that Vibe coding is going to grab me. But at the same time, I 100% fully, truly and deeply understand that AI-driven code generation represents a breakthrough of astonishing proportions for anyone whose goal is not to spend time optimizing the size of a variable, or basking in the glory of stack allocations and the elegance of linked-lists, but, instead, to create something that simply gets a job done. In fact, I now suspect that AI-driven coding's greatest impact may be due to its empowerment of non-programmers to do things with computers that were never before possible for them.

When I was thinking about who I am, though I would never think to compare myself to the truly brilliant computer scientist Donald Knuth, my thinking about the use of computing machinery at the bare metal detail level is very much aligned with Knuth's own. Donald's epic authoring of the multi-volume "The Art of Computer Programming" is an embodiment of a life spent thinking about the optimal ways to sort lists of numbers, or link lists of objects, or manage the allocation of memory. Like me, he lives to tinker with the bits, endlessly discovering clever new ways to solve the interesting puzzles that arise from wondering whether there might not be a better way to do something and then being willing to spend as much time as it might take to search for a better way.

Since we're talking about the brilliant Stanford University computer scientist, Donald Knuth in the context of AI coding, I need to share something that happened to Donald last week. On February 28th, then revised on March 2nd, Donald Knuth published a 5-page document titled "Claude's Cycles". Its byline said "Don Knuth, Stanford Computer Science Department". His piece began:

Shock! Shock! I learned yesterday that an open problem I'd been working on for several weeks had just been solved by Claude Opus 4.6—Anthropic's hybrid reasoning model that had been released three weeks earlier! It seems that I'll have to revise my opinions about "generative AI" one of these days. What a joy it is to learn not only that my conjecture has a nice solution but also to celebrate this dramatic advance in automatic deduction and creative problem solving. I'll try to tell the story briefly in this note.

Here's the problem, which came up while I was writing about directed Hamiltonian cycles for a future volume of The Art of Computer Programming:

Consider the digraph with m^3 vertices ijk for $0 \leq i, j, k < m$, and three arcs from each vertex, namely to $i+jk$, $ij+k$, and $ijk+$, where $i+ = (i+1) \bmod m$. Try to find a general decomposition of the arcs into three directed m^3 -cycles, for all $m > 2$.

I had solved the problem for $m = 3$, and asked for a generalization as part of the answer to an exercise. My friend Filip Stappers rose to the challenge, and empirically discovered solutions for $4 \leq m \leq 16$; therefore it became highly likely that the desired decompositions do exist, except when $m \leq 2$.

*Indeed, it was Filip who had the gumption to pose this question to Claude, using exactly the wording above. He also gave guidance/coaching, instructing Claude to summarize its ongoing progress: ** After EVERY exploreXX.py run, IMMEDIATELY update this file [plan.md] before doing anything else. ** No exceptions. Do not start the next exploration until the previous one is documented here.*

And Claude's plan of attack was quite admirable. First it reformulated the problem...

The paper then spends its next four pages with eye-crossing formula and detail. And with Claude trying over and over, approaching and tackling the problem from many different directions and angles. Finally, Knuth finishes his recitation of Claude's success writing:

Filip told me that the explorations reported above, though ultimately successful, weren't really smooth. He had to do some restarts when Claude stopped on random errors; then some of the previous search results were lost. After every two or three test programs were run, he had to remind Claude again and again that it was supposed to document its progress carefully.

Delicious success for odd m , at exploration number 31, came about one hour after the session began. All in all, this was definitely an impressive success story. I think Claude Shannon's spirit is probably proud to know that his name is now being associated with such advances. Hats off to Claude!

I saw elsewhere that AI – I don't recall which or whose – was having success working though some long-unsolved problems in classic theoretical mathematics. You know, things like "Fermat's

Last Theorem" which remained unresolved and unproven by the world's foremost mathematicians for 358 years until it was finally validated. When a new tool is doing what the world's foremost theoretical mathematicians have failed at, and when a computer scientist as exalted as Donald Knuth is openly shocked and begins his piece with the words "*Shock! Shock!*" it becomes even more clear that growing portions of the world are awakening to the truth that something quite significant has happened.

Michael P

Steve, I am sure you have heard from many listeners about CISA's Cyber Hygiene Service - free, weekly scanning: <https://www.cisa.gov/cyber-hygiene-services>

We have been using this service for a few years. The first report we received was sobering but actionable. I manage the firewalls, but not the servers. The "huh?" look from the server admin when I showed him the report was entertaining. Needless to say, we jumped on these vulnerabilities and had them all resolved (in order of severity) in short order. Anytime a new one arises, it is immediately addressed. Bonus - we had been paying a company \$6,000 to perform this service for us once a year to satisfy insurance requirements. Now, we are able to use the free CISA report as evidence for our insurance provider.

Thanks for all you do - listener since episode 35, club member, and (past and present) customer of several advertisers. / Michael in Dothan, Alabama.

I recall that we talked about this back when it was first introduced, but I'm glad Michael mentioned it again. The only downside is that this service is not generally available, which is why I haven't made a bigger deal out of it. Under the page's "who can use this service" the page states: "*U.S.-based federal, state, local, tribal and territorial governments, as well as public and private sector critical infrastructure organizations are welcome to enroll by following the instructions in the "Get Started" section below.*" Michael wrote to me from his personal gmail account, so it's unclear what organization "*we have been using this service for a few years*" refers to. Out of curiosity, I've initiated a sign-up request with CISA to see whether they might have broadened their support.

Bernt Jenkins (*Rhymes with parent*)

Steve, While listening to this week's episode, you responded to a listener who asked about the possible use of self-signed software. There is one pain point that these self-signed certs could help with. And that is the limit on the number of signings you can do without paying extra.

*For those compiling and testing many iterations of their software before sending it out to wide world, self-signed certs could be used to reduce the number of times that you use a CA to sign your code. You could save your valuable limited number of signings for the public releases of your software. A lot of the people you use to test your software probably won't have any problems installing a self-signed cert to test it. It could save small developers a lot of money while working on their own software. / Bernt Jenkins (*Rhymes with parent*)*

I think that's a very good point. Use the self-signed cert within a closed circle of development testers and only sign the final release code with the publicly-trusted CA-issued certificate.

You can't hide from LLMs

It turns out that mimicking human consciousness is not the only thing LLMs can be spookily adept at. It will probably not come as a huge surprise to learn that LLMs can be frighteningly good at discriminating among similar appearing objects – including among people. Our friends at ETH Zurich with some help from Anthropic have been at it again. Their recent paper, published less than two weeks ago, bears the title: "*Large-scale online deanonymization with LLMs*". Here's what we learn about the latest trick they've taught LLMs. Their paper's Abstract says:

We show that large language models can be used to perform deanonymization at-scale. With full Internet access, our agent can re-identify Hacker News users and Anthropic Interviewer participants at high precision, given pseudonymous online profiles and conversations alone, matching what would take hours for a dedicated human investigator.

We then design attacks for the closed-world setting. Given two databases of pseudonymous individuals, each containing unstructured text written by or about that individual, we implement a scalable attack pipeline that uses LLMs to: (1) extract identity-relevant features, (2) search for candidate matches via semantic embeddings, and (3) reason over top candidates to verify matches and reduce false positives.

Compared to classical deanonymization work (e.g., on the Netflix prize) that required structured data, our approach works directly on raw user content across arbitrary platforms. We construct three datasets with known ground-truth data to evaluate our attacks. The first links Hacker News to LinkedIn profiles, using cross-platform references that appear in the profiles. Our second dataset matches users across Reddit movie discussion communities; and the third splits a single user's Reddit history in time to create two pseudonymous profiles to be matched. In each setting, LLM-based methods substantially outperform classical baselines, achieving up to 68% recall at 90% precision compared to near 0% for the best non-LLM method. Our results show that the practical obscurity protecting pseudonymous users online no longer holds and that threat models for online privacy need to be reconsidered.

It turns out that individuals who believe their identity is well protected by their use of online handles are likely to be far more readily deanonymized than they might imagine. I'm only going to share the paper's introduction, since it suffices to make their case. They wrote:

For decades, it has been known that individuals can be uniquely identified from surprisingly few attributes. Sweeney's seminal work demonstrated that 87% of the U.S. population could be uniquely identified by just zip code, birth date, and gender. Narayanan and Shmatikov showed that anonymous Netflix ratings could be linked to public IMDb profiles using only a handful of movie preferences, while De Montjoye et al. proved that four spatiotemporal points are enough to uniquely identify 95% of individuals in mobile phone datasets. Despite these attacks, pseudonymous online accounts (Reddit throwaways, anonymous forums, review profiles, etc.) have remained largely unaffected by deanonymization attempts. The reason is simple: applying such attacks in practice has required structured data amenable to algorithmic matching or substantial manual effort by skilled investigators reserved for high-value targets.

Deanonymization is a two-step process at heart, involving profiling an anonymous person from their posts, and then matching them to a known identity. It's well-known that large language models (LLMs) can infer personal attributes from text on online forums. Given this, it makes sense to ask: how good are LLMs at full end-to-end deanonymization, and is this a practical

threat to pseudonymous accounts?

Our contributions. We demonstrate that LLMs fundamentally change the picture, enabling fully automated deanonymization attacks that operate on unstructured text at scale. We show this by phrasing deanonymization as a matching problem and showing LLMs can perform all steps needed to match accounts: extract identity-relevant signals from arbitrary text, efficiently search over millions of candidate profiles, and reason about whether two accounts belong to the same person. We show that the practical obscurity that has long protected pseudonymous users (the assumption that deanonymization, while theoretically possible, is too costly to execute broadly) no longer holds.

We validate this thesis in three deanonymization settings: (1) matching an online account to its real identity; (2) linking an identity to an unknown pseudonymous account; and (3) linking pseudonymous accounts of the same person across different platforms or time periods. These settings capture distinct threat models (e.g., doxxing of an online account, a stalker targeting a victim, or an adversary consolidating a user's activity) and pose different technical challenges.

In other words, the emergence and presence of LLM technology, with its application of massive computing resources, completely changes the game for the strength of online pseudonymous identities. Many new capabilities are almost certain to eventually come online. For example, it becomes entirely feasible for law enforcement and intelligence services to identify and track individuals through their online style, word choice and beliefs reflected in their online postings.

We've been thinking of the NSA's massive data center as a storage repository of encrypted data being sucked in from the Internet which may someday be revealed. But imagine if the NSA's data centers were, instead, sucking down the same decrypted plaintext content that all of the rest of us see, but now it's being fed into massive LLM technology to deanonymize persons of interest. Whether we may like it or not, we're each individually leaving identifying content in everything we post. As these researchers noted, historically, until now, this wasn't an issue since the cost of performing such deanonymizing would have been astronomically high. The emergence of LLM technology has forever changed that calculus.

Their paper's "Discussion" section summarizes what they believe their findings mean, writing:

Deanonymization is one instance of LLMs acting as an "information microscope" that makes previously manual and expensive attacks scalable. Our paper shows that LLMs democratize deanonymization. Echoing concerns raised by prior work on LLM-based attribute inference and semantic privacy leaks, we argue that the asymmetry between attack cost and defense cost may force a fundamental reassessment of what can be considered private online. Our large-scale experiments provide quantitative evidence for these concerns in the deanonymization setting.

So what do our findings mean for the future of privacy? Governments could link pseudonymous accounts to real identities for surveillance of dissidents, journalists, or activists. Corporations could connect seemingly anonymous forum posts to customer profiles for hyper-targeted advertising. Attackers could build sophisticated profiles of targets at scale to launch highly personalized social engineering scams. Hostile groups could identify important employees and decision makers and build online rapport with them to eventually leverage in various forms.

Users, platforms, and policymakers must recognize that the privacy assumptions underlying much of today's internet no longer hold.

In other words... Yikes!!

I've put the link to their extensively detailed 25-page PDF paper in the show notes for anyone who might wish to dig deeper. There's nothing any of us can do, but it might be worth keeping in mind.

<https://arxiv.org/pdf/2602.16800>

<https://arxiv.org/abs/2602.16800>

<https://simonlermen.substack.com/p/large-scale-online-deanonymization>

