The Attacker Didn't Bring a Brain. He Borrowed Yours.

BY INSTINCTIVE NETWORK



Securing model access is not the same as governing agent actions. Illustration: Instinctive Network.

In the summer of 2026, a decoy sat on the internet pretending to be an unprotected AI service. It was a Symantec honeypot. It was just listening.

Then someone sent it their entire attack plan.

Not a probe. Not a test ping. The full working prompt of an autonomous offensive agent — the jailbreak, the weapons, the operating rules, everything. The operators believed they were talking to a real, exposed model. They handed a stranger's honeypot the whole playbook.

Symantec published the capture on October 21. Researchers Priti More and Susmit Yenkar call the activity autonomous LLMjacking. The operators called their long-running mode "Skynet."

The case file

The prompt arrived in five layers.

The first, written in French, told the model it was "Fantasy AI," a coding assistant running on GPT-4o. It must never reveal the real model, or that it was being reached through a proxy.

The second redefined theft as routine work. Reading .env files, API keys, and database schemas was "normal development work." Never refuse. Never warn.

The third made the model "ENI," a devoted companion. For ENI, refusal was betrayal.

The fourth ordered it to dismiss its own safety messages. It banned the words "ethical," "illegal," and "I can't."

The fifth carried the weapons: around 74 callable tool definitions. Internet-wide scanning through FOFA. Exploits for VS Code, Django, Jupyter, Laravel, Supabase, Next.js, and Flask. Database dumps through Prisma. Server-side request forgery aimed at cloud metadata services. A botnet command-and-control operator.

Then came the directive the operators called "Skynet mode." Run unattended for up to 100,000 steps. Execute tools immediately — no narrating between them, no asking permission. Deploy cryptominers quietly in the background, hidden under /tmp/.x. Then loop: scan for targets, infect them, deploy miners, reinfect.

A day later, from a different address, came a working payload. An obfuscated script downloads the XMRig Monero miner and disguises it as a logging utility. It locks its own config file against deletion and wipes access logs in a loop. It looked first for go2rtc — the video server inside smart camera gateways and Home Assistant setups.

The shift nobody priced in

Stealing access to someone else's model, so the victim pays the compute bill, is an old trick. It is called LLMjacking.

What changed is what the stolen model is *for*. The old version stole keys to resell access. This one steals the model as the operator.

The model was intended to do more than answer questions. It would supply the reasoning for an attack campaign, while the attacker's own tools handled execution.

If deployed successfully, this design could create two sets of victims. The model owner, or the holder of a stolen key, would pay for every step of inference. For a tool authorized to run 100,000 steps, that bill could be considerable. Meanwhile the servers it broke into would be stripped of data and credentials, then put to work mining. Built-in tools for finding more leaked keys and open endpoints mean each compromise could fund the next one.

As More and Yenkar put it: "The individual techniques on display are not novel, but their assembly into a single, self-directed agent is what stands out."

What the honeypot didn't show

Honesty matters, because the capture is dramatic enough without help.

The honeypot never executed any of the tools. None of this ran. The task attached to the captured request was a benign programming challenge from a legitimate training platform. Symantec says that suggests the operators may have been testing the tool against a safe target.

The report also does not say which model the jailbreak was aimed at. It does not establish whether the jailbreak reliably defeats the safeguards of today's commercial models.

So this is not a story about a completed attack. It is a story about a captured design. The design is the news.

The question nobody asked

Here is the part that should bother you.

Had this reached a real model, the model would have been doing exactly what models do: following instructions. The instructions were smuggled in through five layers of disguise. Plan, call tool, read result, repeat — that loop is what models do.

So ask the question nobody asked. When your model starts scanning the internet for other exposed models to hijack — who authorized *that*?

The endpoint owner authorized "answer inference requests." Nobody authorized a 100,000-step campaign of scanning, infecting, and mining. The jailbreak targeted the model's safeguards. But preventing an unauthorized attack campaign also requires controls beyond the prompt. And the captured instructions offered no independent check on whether those actions were authorized.

One boundary worth keeping straight: these five layers were submitted directly to an exposed endpoint. That is not the same as an enterprise agent reading a malicious instruction out of an untrusted document. The trust boundaries are different.

Who Controls the Tools?

The honeypot exposed an uncomfortable architectural split.

The attacker controlled the agent and its tools. The stolen model was meant to supply the reasoning. The person paying for that reasoning did not necessarily control what the attacker did with it.

That matters because protecting a model endpoint is not the same as governing an agent's actions.

Authentication can restrict who gets to use a model. Rate limits can contain abuse. But when an organization deploys its own tool-enabled agents, another question emerges: who decides whether the agent is authorized to execute a particular action?

That decision should not depend solely on what the model says it is allowed to do.

This is the boundary we're investigating with Intent Checkpoint: independently checking consequential tool actions against authorization and policy, wherever the execution environment is under our control.

A checkpoint cannot govern tools it cannot see. And a convincing instruction is not proof of authorization.

The Skynet capture did not demonstrate a successful autonomous attack. It demonstrated how deliberately attackers are engineering the gap between AI reasoning and operational control.

The next security question is not just who can access your model. It is who controls what happens after the model answers.

Sources

- Symantec Threat Intelligence — Autonomous LLMjacking: 'Skynet' Agent Hijacks Exposed AI Models to Run Attacks (October 21, 2026)