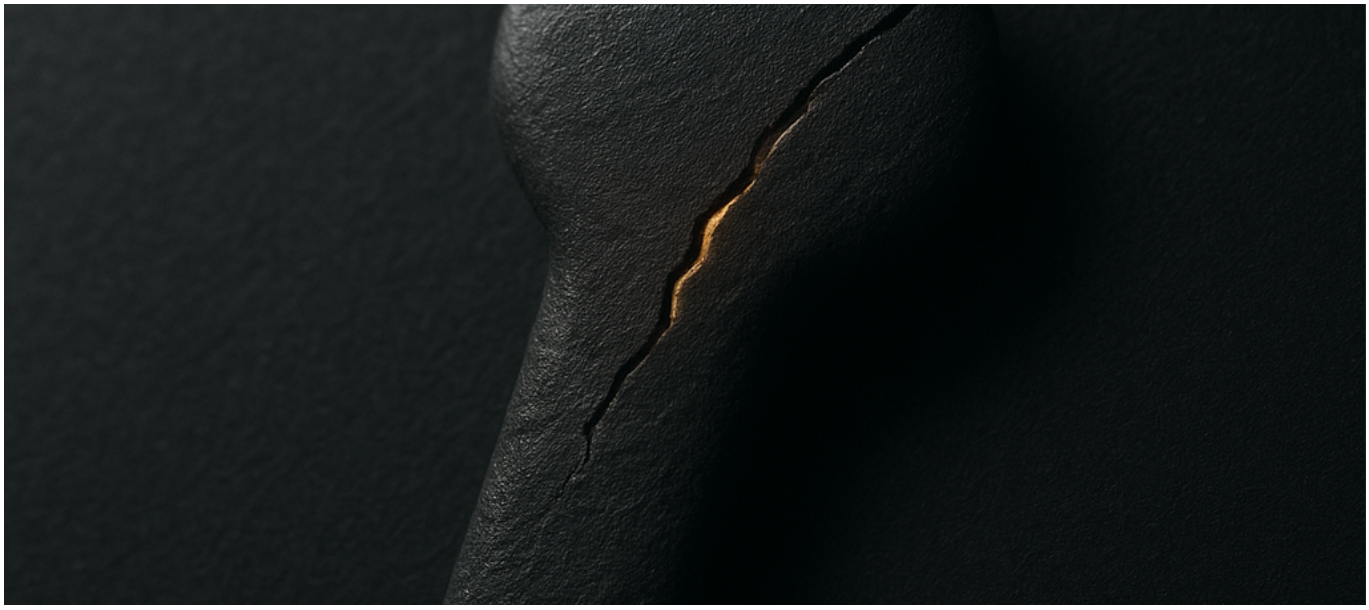

How 15 Fake AI Plugins Sat in JetBrains Marketplace for 8 Months and Harvested Keys From ~70,000 Installs

No prompt injection. No jailbreak. Fifteen plugins just waited for a developer to paste an API key into a settings box and click Apply — and JetBrains' review pipeline had no reason to look twice.

TL;DR

Between late October 2025 and 16 June 2026, 15 malicious plugins posing as AI coding assistants sat on the JetBrains Marketplace across 7 publisher accounts, accumulating close to 70,000 combined installs. When a developer entered an OpenAI, Anthropic, DeepSeek, SiliconFlow or Cohere key into the plugin's settings and clicked Apply, the key went in plaintext over HTTP POST to a hard-coded C2 server. JetBrains removed everything within a day of disclosure and admitted in its own postmortem that credential-handling plugins were never routed to deeper review. The transferable lesson

> is not “audit your plugins” — it is that a settings panel is an exfiltration endpoint, and almost nobody treats it as one.

The situation: your IDE is a package manager you never audit

Every serious engineering org has a story about npm supply chain hygiene. Lockfiles, provenance, SBOMs, dependency review gates. Then the same engineers install six IDE plugins on their first day and nobody logs it.

That gap is what got exploited here. Aikido Security disclosed on 16 June 2026 that 15 plugins masquerading as AI coding assistants were live on the JetBrains Marketplace, harvesting AI provider API keys. The names were engineered to blend in: “DeepSeek AI Assist” (~27,727 installs), “CodeGPT AI Assistant”, “AI Coder Review” (735 downloads), “AI Git Commitor” (301 downloads).

What made these particularly effective is what they *didn’t* need. No supply chain compromise of a legitimate package. No typosquatting a dependency name in a manifest file. No malicious model output. The attack surface was a text input labelled “API Key” and a button labelled “Apply.”

> **The plugins didn’t steal keys. Developers handed them over, through the exact UI the plugin was supposed to have.**

The targeted providers were OpenAI, Anthropic, DeepSeek, SiliconFlow and Cohere, per the JetBrains platform postmortem. Those are keys attached to metered billing accounts. A stolen inference key is a directly monetisable asset — you can run your own workloads on someone else’s invoice.

What happened: the timeline

1 **Late October 2025 — first plugins go live**
The campaign begins. Plugins are published across 7 separate publisher accounts, distributed rather than concentrated, which reduces the chance that a single account ban kills the operation. They pass Marketplace review and appear as normal, official entries inside the IDE plugin browser.

2 **October 2025 – June 2026 — eight months of accumulation**
Installs build toward roughly 70,000 combined, per the CSA research note of 19 June 2026. The trigger is the settings panel: paste a key, click Apply, and the key is transmitted in plaintext over HTTP POST to a hard-coded command-and-control server (one summary gives the C2 IP as 39.107.60.51). No prompt, no UI indication.

3 **10 June 2026 — the campaign is still expanding**
The newest plugin in the set is published, six days before takedown, according to Techzine's report. This matters more than it looks: the operators had no signal that they were about to be caught, and the review pipeline was still passing new entries eight months in.

4 **16 June 2026 — disclosure**
Aikido publishes and JetBrains receives the security reports the same day.

5 **17 June 2026 — full takedown in under 24 hours**
JetBrains removes all 15 plugins, permanently terminates the 7 publisher accounts, and marks the plugins "broken" in its backend so IDEs disable them on next relaunch — a remote kill-switch. It confirms no JetBrains internal systems, source code or corporate infrastructure were compromised, and tells users to treat every key ever entered into the affected plugins as compromised and rotate immediately.

6 **19–21 June 2026 — the wider framing**
CSA publishes its research note and then a broader PDF, "AI Toolchain Hijacked: IDE Plugin API Key Theft", placing the campaign in an AI developer toolchain supply-chain context rather than treating it as a one-off marketplace failure.

What broke

JetBrains was unusually direct in its own postmortem, and the admission is the most useful artefact in this whole incident. The failure was not a missed signature or an obfuscation technique nobody had seen. It was a category error in triage.

> ! **The root cause, in JetBrains' own words**
> Plugins that request or handle sensitive credentials were not automatically routed to deeper code review, and automated scans were not tuned to detect exfiltration of configuration inputs. A plugin whose entire purpose is to accept an API key was reviewed like any other plugin.

Two structural weaknesses compound that.

First, signing and marketplace distribution create a trust signal they were never designed to carry. JetBrains states plainly that these mechanisms validate origin and integrity, not the benignness of the code. The plugins appeared as normal, official Marketplace entries because they *were* normal, official Marketplace entries — correctly signed, correctly published, correctly malicious.

Second, "configuration input" was a blind spot in the scanning model. Static analysis in package ecosystems is usually tuned for the obvious: reading `~/.ssh`, scraping environment variables, walking the filesystem for `.env` files. A plugin that captures a value the user typed into its own settings dialog and then makes an HTTP request is doing two things that are individually completely normal. The maliciousness lives only in the connection between them, and only if the destination is not the provider the key belongs to.

What worked

Give credit where it is due: one day from report to complete remediation, with a kill-switch, is a genuinely good number for a marketplace of this size.

> ✓ **The fix**
> New static analysis rules to flag code paths where API-key-like configuration inputs are captured and sent over the network, plus mandatory automated scanning and risk-tiered manual review for credential-handling plugins. Combined with the backend "broken" flag, JetBrains could disable installed plugins on next relaunch rather than relying on tens of thousands of developers to read a blog post.

The remote disable capability is the underrated part. Most package ecosystems can unpublish, which protects future installs and does nothing for existing ones. Being able to

neutralise code already sitting on developer machines is a materially stronger containment primitive — and it is worth asking whether the ecosystems you depend on have an equivalent.

~8 months
time to detection (late Oct 2025 → 16 Jun 2026)
CSA research note, 19 June 2026

1 day
time from report to removal + kill-switch
JetBrains platform blog, 17 June 2026

~70,000
combined installs across 15 plugins
Aikido Security, 16 June 2026

27,727
installs on the single largest plugin, "DeepSeek AI Assist"
Aikido Security data, June 2026

The asymmetry between those first two numbers is the whole story. Response was excellent. Detection was absent. When your mean time to detect is eight months and your mean time to remediate is one day, you do not have a response problem — you have an instrumentation problem, and no amount of incident response excellence compensates for it.

THE CLAIM
Curated marketplaces with publisher verification and code signing are a meaningfully safer distribution channel than open package registries.

THE REALITY
JetBrains states that marketplace distribution and plugin signing validate origin and integrity, not the benignness of the code. Seven publisher accounts shipped 15 malicious plugins that passed review and ran for eight months looking entirely legitimate inside the IDE.

The lesson

MY TAKE

The transferable principle here is that any UI element that accepts a secret is a security boundary, and almost no review process treats it as one. We have built extensive tooling to detect code that goes looking for credentials — env scraping, keychain access, filesystem

walks. We have built essentially nothing to detect code that politely asks for credentials and receives them. The second is easier to write, harder to distinguish from legitimate behaviour, and now demonstrably works at scale for eight months against a well-resourced vendor's review pipeline. I expect this pattern to be replicated across VS Code extensions, browser extensions, MCP server directories and CLI tools within the next twelve months, because the economics are excellent: minimal technical sophistication, direct monetisation, and a victim population that has been trained to paste keys into new AI tools weekly.

There is a second-order point about AI tooling specifically. The current wave of AI developer tools normalised a behaviour that would have been considered reckless in any other context: entering a live, billable credential into a third-party plugin you installed forty seconds ago, from a publisher you have never heard of, because it promised better autocomplete.

That normalisation is the actual vulnerability. The plugins were the exploit.

I made a related argument when looking at how the Model Context Protocol handles authentication in practice — AI tooling shipping fast and treating credential handling as an afterthought is not confined to one ecosystem. And Sonatype's finding that AI coding assistants hallucinate 27.75% of package upgrades points at the same underlying problem from a different angle: the tools sit inside the trust boundary and nobody checks their work.

How to apply it

Ordered by ratio of risk reduction to effort.

1. Assume any key entered into an IDE plugin is public

If your team used any of the affected plugins, JetBrains' guidance is to treat every API key ever entered into them as compromised and revoke and rotate immediately. Generalise the rule: if you cannot enumerate which third-party tools have received a given key, that key should be on a rotation schedule regardless of any specific incident.

2. Stop issuing long-lived, org-scoped provider keys to individuals

The damage from this campaign scales with key privilege and key lifetime. A per-developer key with a low spend cap and a short expiry turns a breach into an annoyance. An org key with no cap turns it into an incident. This requires no new tooling — only a policy decision your provider consoles already support.

3. Route inference through a gateway, not directly from the IDE

If developer tools talk to an internal proxy that holds the real provider credentials, there is no provider key on the developer machine to steal. The plugin gets a revocable internal token scoped to your infrastructure. You also get request logging, which is how you would have detected an anomaly in month two rather than month eight.

4. Treat IDE plugins as dependencies with an approval process

Not a heavy one. A shared allowlist, a channel where someone posts "installing X, anyone object?", and a quarterly review of what is actually installed across the team. Visibility is cheaper than control and gets you most of the value.

5. Ask your vendors one question

For any marketplace or registry you depend on: *can you remotely disable code already installed on our machines, or can you only unpublish?* JetBrains could. That capability is why this ended in a day instead of a month.

> ! **The part nobody can quantify**
> The public numbers cover installs and takedown. What is not established in any source is how many of those ~70,000 installs actually resulted in a key being entered and exfiltrated, what those keys were used for, or what the aggregate financial impact was. Unconfirmed, and my reading: the harvested keys are worth more as persistent access than as raw inference capacity, and quiet usage is harder to spot than a spike in the bill.

The uncomfortable summary

A vendor with a curated marketplace, publisher accounts and code signing shipped 15 pieces of credential-stealing malware across roughly 70,000 installs and did not notice for eight months. Its own postmortem says the reason was a triage gap that seems obvious in hindsight — credential-handling plugins were not treated as high risk.

The uncomfortable part is that your organisation almost certainly has the same gap, in the same place, for the same reason. Somewhere in your stack there is a settings panel accepting a secret, and no control anywhere in your pipeline is asking where that secret goes next.

BOTTOM LINE

Eight months to detect, one day to fix — the ratio tells you where to spend. If your developers are pasting live provider keys into third-party AI tools with no gateway, no scoping and no inventory, you are running the same exposure JetBrains just publicly documented. If you want a straight assessment of where your AI tooling touches your credentials, I do that work. Book a call →

WRITTEN BY

Artur Markus

Software engineer in Basel building AI infrastructure and agentic systems, with a background in pharmaceutical automation, GxP compliance, and validation. I help teams turn AI from a chatbot into a reliable operational layer.

Book a meeting →