

Incident Response for AI-Related Leaks

For security and leads · ~12 min read

At some point, somebody on your team is going to paste something they should not paste into a tool they should not use. They will realise it three seconds after pressing enter, or three minutes later in the shower, or three days later when a customer asks an awkward question. The moment that happens, you are no longer doing prevention — you are doing incident response. This lesson is the playbook for the hour that follows.

AI-related leaks are not a new category of incident, but they have their own texture. The data leaves over HTTPS to a provider you may have no contract with. The "system" that received it is a chat interface that does not expose server logs to you. The provider may keep the data for thirty days, or forever, or until they next train a model on it, and the paperwork is quieter than you would like. Your job is not to argue with that landscape. Your job is to contain, assess, rotate, notify, and document — in that order — fast enough that the exposure has a boundary.

The 5-step playbook

1. Contain

Contain is the step everyone wants to skip because it feels passive. It is not. Contain means two things at once: stop the bleeding, and freeze the scene. Stop the bleeding is obvious — tell the person to stop using the tool, close the prompt box, do not send another message, do not "just ask one follow-up to see what it says." Freezing the scene is the part people forget. Before any credential is rotated, before anybody is notified, before the adrenaline takes over, you need a captured record of exactly what was sent, exactly when, from which account, over which network. Screenshots of the prompt and response. A verbatim copy of the prompt text into secure notes. The URL, the account identifier, the browser time zone. Do not delete the chat. Deleting the chat destroys your own timeline and does not remove the data from the provider, so it is pure downside. The tab stays open.

2. Assess exposure

Once the scene is frozen, figure out what was actually exposed. This is more than re-reading the prompt. Assume the provider saw everything you sent them — the text, yes, but also the metadata: the account, the IP, the timestamp, the browser fingerprint, the prior messages in the same chat session, the system prompts the client injected on your behalf. Then ask what the provider's retention policy is on the specific plan the user was on. Consumer plans and enterprise plans often differ, and a "zero retention" promise on one tier does not apply to the other. Then ask who else sees the data: the provider's own operations team, abuse-review contractors, model-training pipelines, hosting sub-processors, logging services. Finally, ask who else on your side could reach this chat — a shared account, a team workspace, history sync to a second device. The output of this step is a sentence of the form "N pieces of data of type T were exposed to provider P under plan Q, with retention R and sub-processors S." Write it down.

3. Rotate

Rotation is where people make the classic mistake of doing only "the obvious" credential. The rule is the whole chat session is the blast radius, not the single message the user remembers. Scroll up. Every secret, every token, every path, every identifier that appeared anywhere in that session is in scope. The priority order, when time is short, is: session tokens first (they are already active, already usable, and cannot be un-sent); then API keys (because they are long-lived and scriptable); then passwords (because humans reuse them and an attacker with the password gets everything the human has); then long-lived infrastructure credentials — DB connection strings, signing keys, IAM access keys, service account JSONs — because those are the ones whose blast radius inside your systems is largest. Rotate, revoke the old value, and check audit logs for any use of the old credential between the paste time and the rotation. Unfamiliar use is an escalation.

4. Notify

Notification happens on a clock, and the clock starts at T+0, not when you "feel ready." Internal security gets told first, within minutes. The manager or lead who owns the affected system gets told next — they need to know before any external party does, not after. If personally identifiable information was in the chat context, legal and privacy get looped in, because a regulatory reporting clock may already be running and late notification is its own, separately punishable incident. If your customer contracts

require breach notification for this kind of exposure, the contract SLA is the deadline, not a target. Document who you notified, when, and what you told them; a notification you cannot prove happened might as well not have happened.

5. Document

The final step is a blameless postmortem. Blameless does not mean vague. It means the root cause you write down is a systemic one, not a person. In the great majority of AI leak incidents the real root cause is the same shape: the tool was approved for some uses, but the training and the guardrails for the kinds of data it must not see were not. The engineer knew ChatGPT existed and was allowed. They did not know that the production DB string they were debugging was in the "never paste" category, because nobody had ever drawn that line explicitly. The postmortem fix is therefore a policy or tooling change — a DLP rule, an approved-tool list with data-type annotations, a redacting proxy, a training update, a clipboard warning — and not a note on the engineer's file.

The third-party training endpoint case

The single most common AI-specific incident is the one where an employee, usually on their own device, pastes corporate data into a consumer AI account. This case deserves its own section because it has properties that do not apply to most other incidents: the provider has no contract with your company, the account is personal, there is no enterprise logging, and the "deletion" story is contractual rather than technical.

When you ask the provider to delete the data, you are asking them to honour their terms of service, not invoking a technical primitive. Most serious providers offer a deletion-request form and will process it. None of them give you a proof-of-deletion you can hand to an auditor. Some of them exclude data that has already flowed into a training batch. Some have sub-processors whose deletion semantics differ from the primary service. And even if every one of those paths works perfectly, you still have no way to verify the data is gone from any backup the provider did not tell you about.

The questions worth asking the provider are specific: what exactly was retained, on which retention schedule, whether it entered any training pipeline, which sub-processors received it, and what the deletion SLA is. File the request. Keep the reference number. Then treat the data as permanently exposed anyway, and let your

rotation and notification plan do the real work. A deletion request is a contractual courtesy. It is not a technical guarantee, and your incident response must not depend on it.

If your plan quietly assumes "we'll just ask them to delete it and move on," you do not have a plan. You have an optimistic wish wrapped in a form submission.

First-hour timeline

A usable timeline for the first sixty minutes, annotated: at **T+0** you acknowledge to the reporter, instruct them to freeze the tab, capture screenshots, and stop using the tool. At **T+2** you identify the specific secret, system, and scope. At **T+5** the primary credential is rotated in the admin console. At **T+8** you pull the audit log for any use of the old credential since the paste time. At **T+12** you rotate everything else that appeared in the same chat session — not just the one the reporter remembers. At **T+20** security lead and the owning manager are notified. At **T+30** the timeline doc exists and a postmortem owner is assigned. At **T+45** legal / privacy / customer notifications go out if required. At **T+60** you submit the provider deletion request, knowing it is a courtesy, and file the incident ticket. An hour is not a lot of time, and this timeline is aggressive on purpose: the alternative is that the first hour becomes the first day and the attacker is no longer theoretical.

Blameless postmortem

The postmortem is where the organisation either learns from the incident or repeats it. The single most common failure mode is stopping at "the engineer should have known better." The engineer probably did know better, in the abstract, and was under time pressure, and was debugging, and reached for the tool that made the pain stop. The systemic question is why the environment made that tool the easiest answer for that kind of data in the first place. The answer is almost always that the tool was approved in general but the training — the specific, concrete "these data types never go into this tool" rule — was not drawn. Fix that. Add the data-type restriction to the approved-tool list. Add a DLP rule that catches connection strings, API keys, and known secret patterns on their way to known AI endpoints. Update the training so the next engineer

does not have to re-derive the line themselves under pressure. Then hold the postmortem review, confirm the systemic fix is scheduled, and close the incident.

A good postmortem answers two questions: what do we change so this particular sequence of events cannot happen the same way again, and what did we learn about the category of risk that applies to other systems. If your doc only answers the first one, you are doing incident clean-up, not incident learning.

Rules to remember

- Contain first. Capture evidence before anything else. Do not delete the chat.
- Assume the provider saw everything you sent, including metadata, including sub-processors.
- Rotate priority: session tokens > API keys > passwords > long-lived creds. All of them, not just the "obvious" ones.
- Notify within your documented SLA. Late notification is its own incident.
- The postmortem is about systems, not people.
- Third-party deletion requests are a contractual courtesy, not a technical guarantee.