PolySwarm for Sekoia.io

User guide for the PolySwarm automation module

PRODUCT	PolySwarm automation module for Sekoia.io
MODULE VERSION	1.0.0
PLATFORM	Sekoia.io Defend and Intelligence Center
AUDIENCE	SOC analysts and playbook authors
SUPPORT	customersuccess@polyswarm.io
PUBLISHED	2026-11-13
CLASSIFICATION	Public

What this is

PolySwarm answers one question: has anything in the industry seen this file, hash, URL, domain or address before, and what did it say. 35+ detection engines assert on each artifact, and this module brings those answers into Sekoia.io so a playbook can act on them.

It installs as an ordinary Sekoia integration. Once an account is connected, sixteen actions and one trigger are available to any playbook, with no separate deployment and nothing to run yourself.

This module is for SOC analysts and playbook authors who want a second opinion on an indicator without leaving Sekoia. It solves three problems. First, an alert arrives carrying a hash, a URL or an address and nobody knows whether it is already known bad; a read-only lookup answers that in one step and costs no scanning quota. Second, an artifact is genuinely unknown and somebody has to decide whether to detonate it; the module submits it and hands the sandbox report back. Third, a verdict is needed on many indicators at once, as part of triage rather than one at a time.

It is deliberately careful about one thing: a result that nobody answered is never reported as clean. That distinction is the difference between “we looked and found nothing” and “nobody looked”, and it is wired through every action.

Key Takeaways

- **Searching is free, scanning is not.** The search actions read what PolySwarm already holds and spend no scanning quota. The scan and detonate actions submit the artifact and spend against your plan.
- **Unknown is not the same as not detected.** Every lookup separates the two, so a playbook can route an unseen artifact to a detonation step instead of treating it as safe.
- **What you submit is shared.** A file or URL sent to a scan or a detonation is visible to every engine in the community your key belongs to, and cannot be withdrawn.
- **Detonation is too slow to wait on.** A fresh sandbox run takes minutes: one URL detonation we timed finished in about four and a half, and a large file takes longer. Use the completion trigger rather than blocking a playbook on it.

Requirements

You need a PolySwarm API key and the name of the community that key belongs to. Everything else is already in Sekoia.

The module runs on the Sekoia platform as a standard integration, so there is nothing to install on your own infrastructure, no agent, and no network path to open from your side.

Install and configure

Find your API key

1. Sign in at `polyswarm.network`.
2. Click the gear icon in the top bar, then `Settings`, then the `API Keys` tab.
3. Keys issued to a team are listed under `Team Account API Keys`, with the community each one is scoped to in the `Community` column. Keys issued to you personally are under `Personal Account API Keys`.
4. Use the copy icon beside the key you want. The key stays masked on screen, so copy it rather than trying to read it.

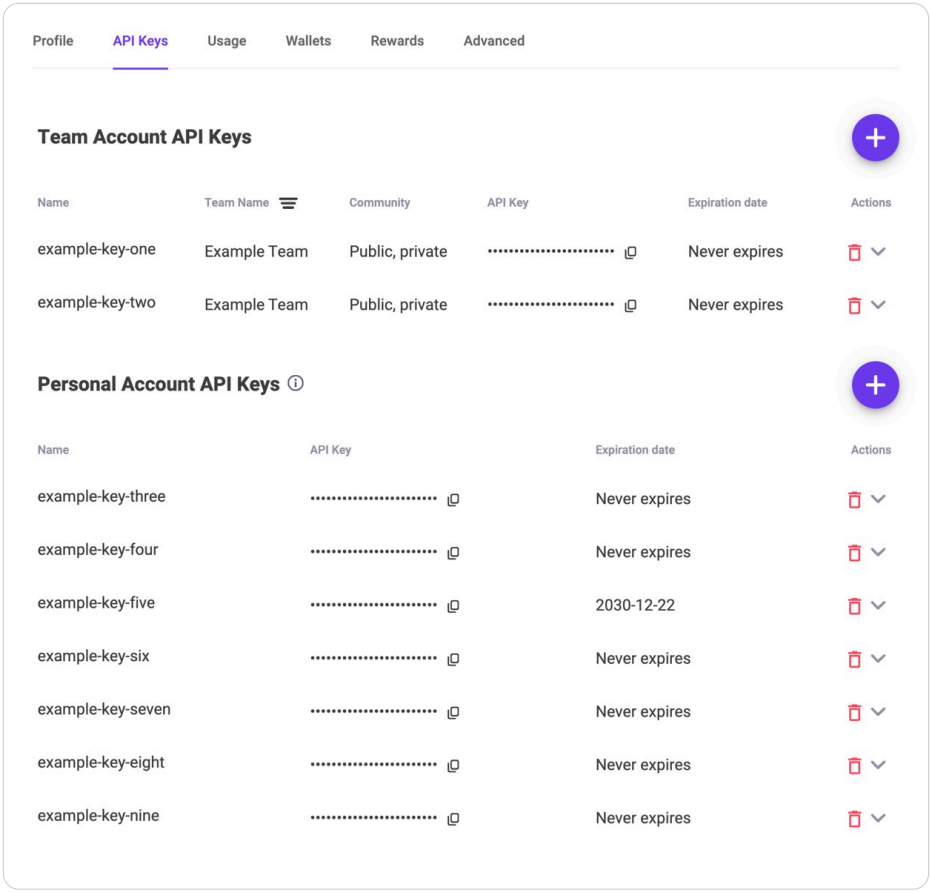

Figure 1: The API Keys tab in the PolySwarm portal. Keys stay masked, so use the copy icon.

If you do not have a key, the round + button on either section creates one, or your PolySwarm contact can issue one for your team.

Connect the account in Sekoia

5. Go to `Configure`, then `Integrations`, and search for `PolySwarm`.

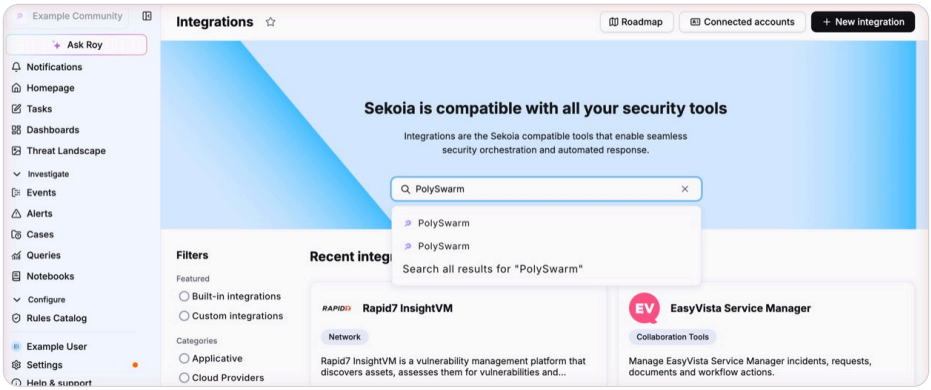

Figure 2: Searching the Sekoia integrations catalogue for PolySwarm.

6. On the PolySwarm page, click `Show accounts`, then `+ Connect an account`.

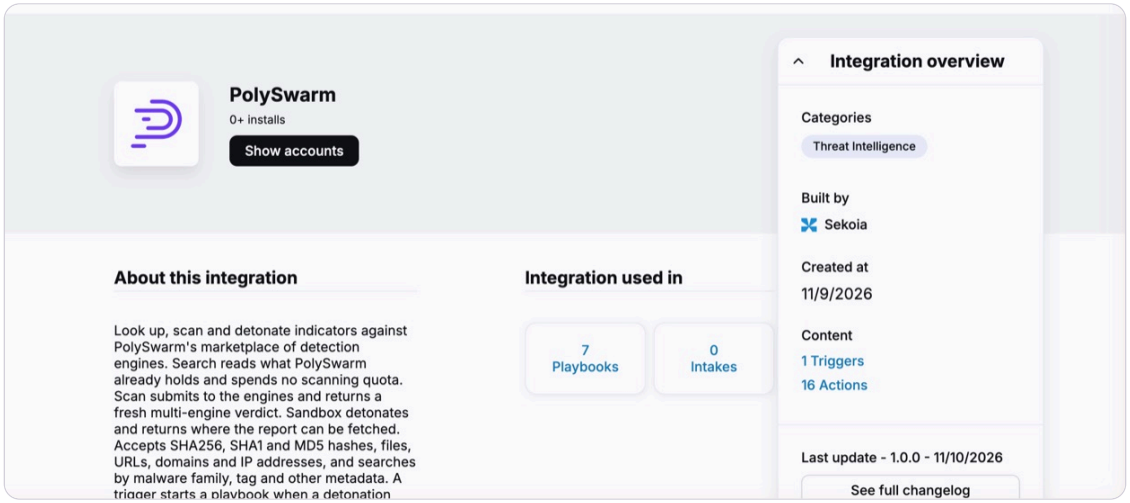

Figure 3: The PolySwarm page. The overview panel on the right carries the version and the content count, and Sekoia publishes the module so it reads “Built by Sekoia”.

7. Give the account a name you will recognise in a playbook, for example the team the key belongs to.
8. Paste the key into `Api Key`, whose placeholder reads `Type in API_Key`. The field is declared as a secret, so the platform masks it once saved.
9. Set `Community`. It is prefilled with `default`, which is the public community and is where most keys belong. If PolySwarm issued you a key for a private community, put that community’s name here instead.

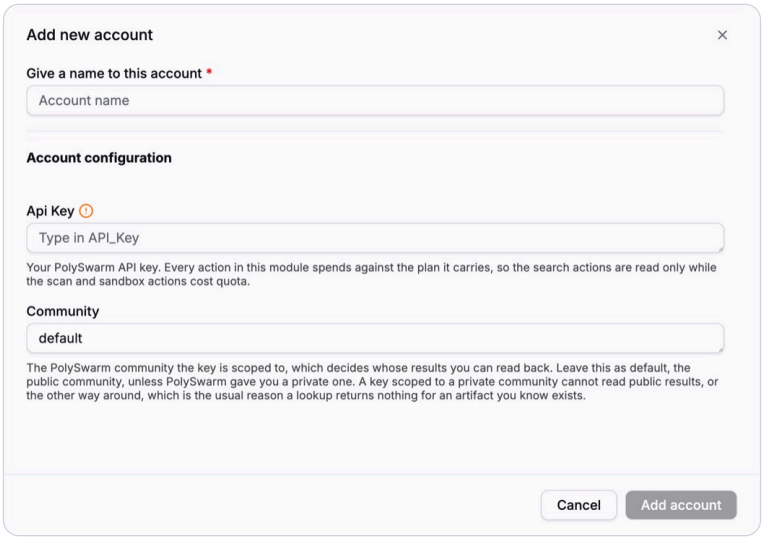

Figure 4: The Add new account form, before anything is typed.

10. Click `Add account`. It then appears under `Connected accounts` and can be selected by any PolySwarm action or trigger in a playbook.

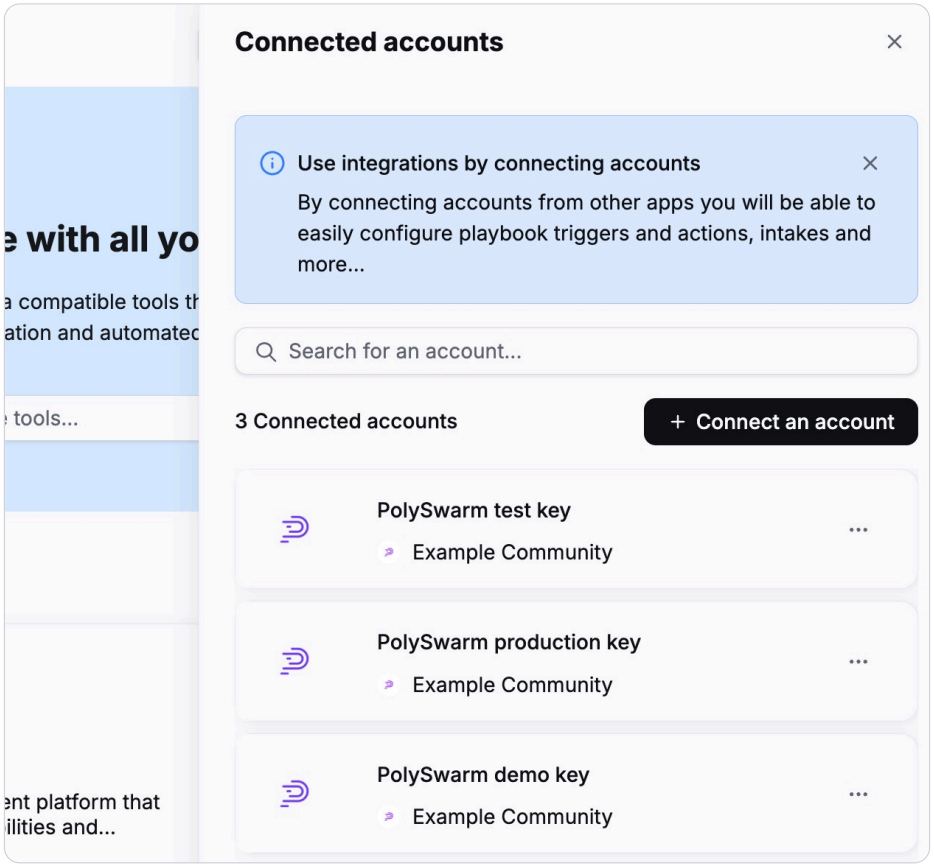

Figure 5: Connected accounts. Each one carries its own key and community.

A word about the community field, because it is the one thing the two products do not line up on. The PolySwarm portal describes a key's community in words, as public or private, and never shows the exact name Sekoia expects here. If lookups come back empty for artifacts you are certain exist, this field is almost always the reason, and your PolySwarm contact can tell you the exact value.

The two settings on the account

Setting	Default	What it does
`Api Key`	`none`	Your PolySwarm API key. Shown on the account form as `Api Key`, with `API_Key` in the placeholder. Declared as a secret, so the platform masks it after you save. Every action spends against the plan this key carries.
`Community`	`default`	The PolySwarm community the key is scoped to, which decides whose results you can read back. A key scoped to a private community cannot read public results, or the other way around.

Adding an action to a playbook

11. Open a playbook under `Configure`, then `Playbooks`, and click the `+` on the canvas.
12. Search for `PolySwarm`. The `Actions` tab lists every action, and the `Triggers` tab holds `Sandbox detonation completed`.

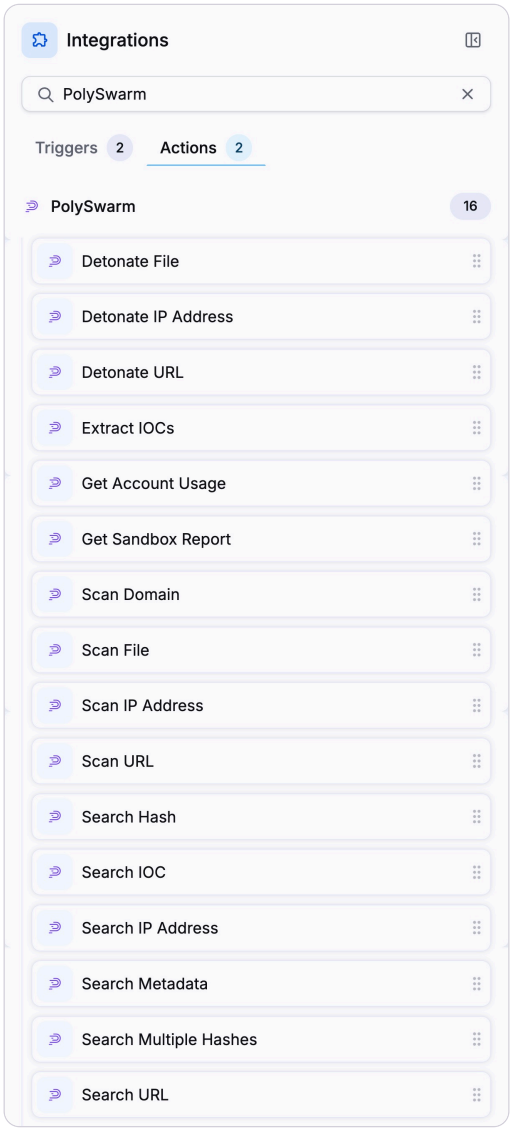

Figure 6: All sixteen PolySwarm actions in the playbook editor.

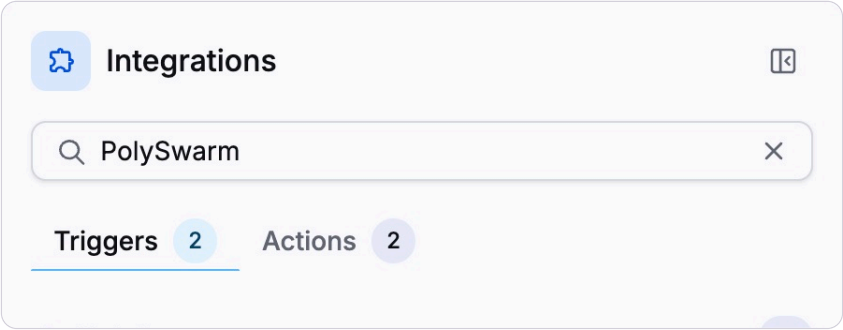

Figure 7: The sandbox completion trigger, on the Triggers tab.

Use cases

Each section below is one job an analyst actually does, the exact steps, and what comes back.

Check a file hash against the industry

The commonest job: an alert carries a hash and you want to know whether anything has seen it. This is read only and spends no scanning quota, so it is safe to run on every alert.

Add `Search Hash` to the playbook and set `query_hash` to the hash from the alert. MD5, SHA1 and SHA256 are all accepted.

The action takes one of three branches. `detected` fires when at least one engine called it malicious, `not detected` when engines answered and none did, and `unknown` when PolySwarm has never seen it or a scan has not finished. Wire all three. The response carries the malicious, benign and total detection counts, the PolySwarm score, the malware family where one is known, and a permalink.



Figure 8: Search Hash on a known-malicious file, with the engine detection counts.

The three counts will not always add up, and that is deliberate. The total is how many engines answered at all; the malicious and benign counts are how many committed to a verdict. An engine can be asked and abstain, and an abstention is counted as neither. A real example: a sample we tested came back 7 malicious and 6 benign out of 14, because the fourteenth engine abstained. Counting that abstention as benign would quietly tilt every ratio a detection rule is written against.

The `unknown` branch is worth dwelling on, because it is the one most likely to be wired wrongly. It does not mean clean. It means nobody has answered yet.

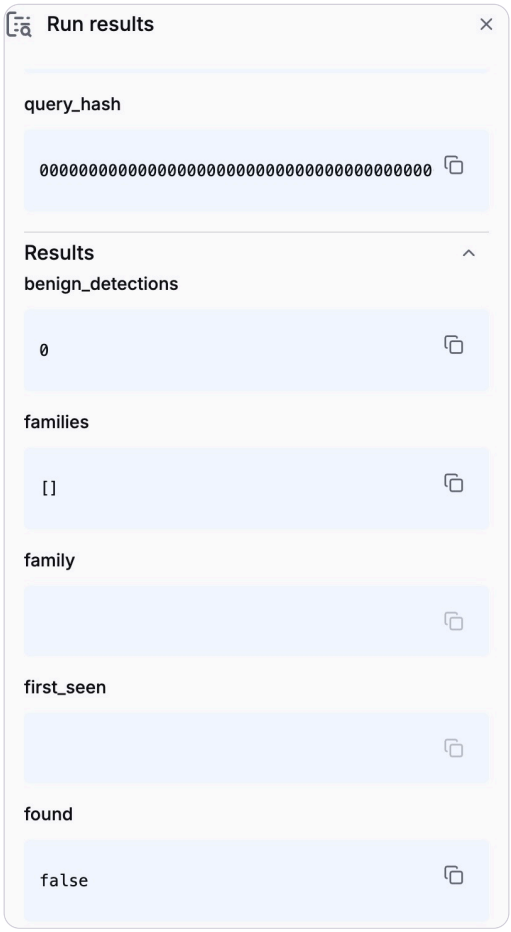

Figure 9: Search Hash on an artifact PolySwarm has never seen, taking the unknown branch.

Check a list of hashes in one step

Use `Search Multiple Hashes` when triage hands you many hashes at once. Set `hashes` to the list. Turn on `include_assertions` only if you need the per engine detail, because it makes the response much larger.

Be aware this is a loop, not a batch: PolySwarm has no bulk hash endpoint, so fifty hashes cost fifty requests. Duplicates are charged once. If the run stops early, for a spent quota, a refused key, an unreachable platform, several failures in a row, or a ten minute time bound, it returns the rows it already gathered and reports `partial`. Treat `partial` as an incomplete answer, never as a clean one for the hashes it never reached.

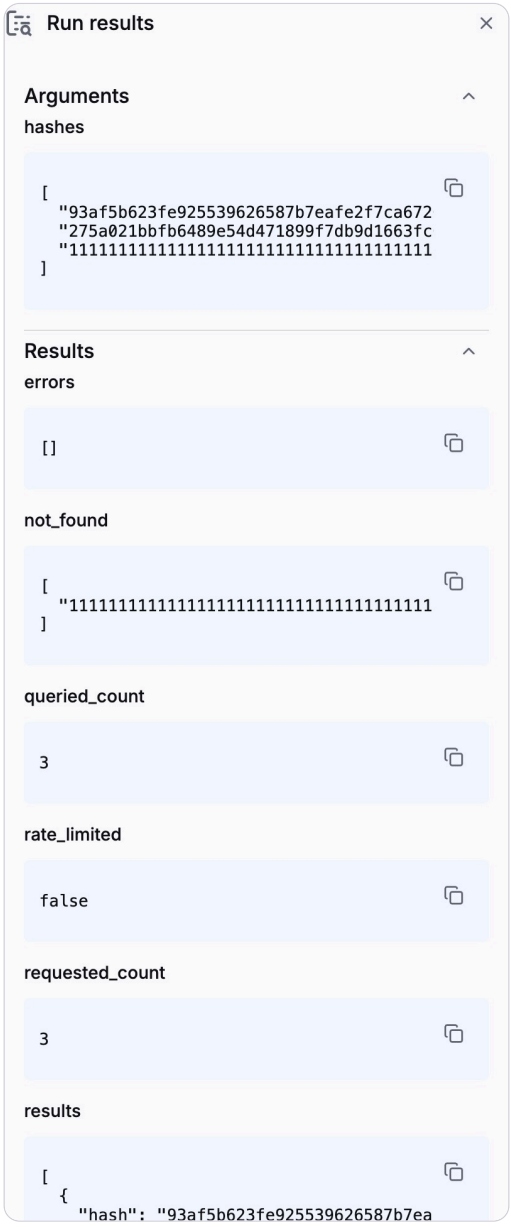

Figure 10: Search Multiple Hashes over a short list, with one row per hash.

Check a URL, an address or a domain

`Search URL` and `Search IP Address` work the same way as the hash lookup and are equally free. Set `url` or `ip`, and read the same detection counts back.

Expect plenty of ordinary indicators to come back not found. That is not a failure: it means nothing in the community has scanned that address or URL, and it is the same `unknown` outcome described above. Of two we checked while writing this, a public resolver address returned engine results and a well known example domain returned nothing at all.

For these, use the detection counts rather than the score. The score carries little information for URLs, domains and addresses; for files it is meaningful across its full range.



Figure 11: A URL or address reputation lookup and its result.

Pivot from one indicator to related artifacts

`Search IOC` goes the other way: give it an address, a domain, a technique identifier or an import hash, and it returns artifacts associated with it. Set whichever one of `ip`, `domain`, `ttp` or `imphash` you have, and cap the answer with `limit`, which defaults to twenty.

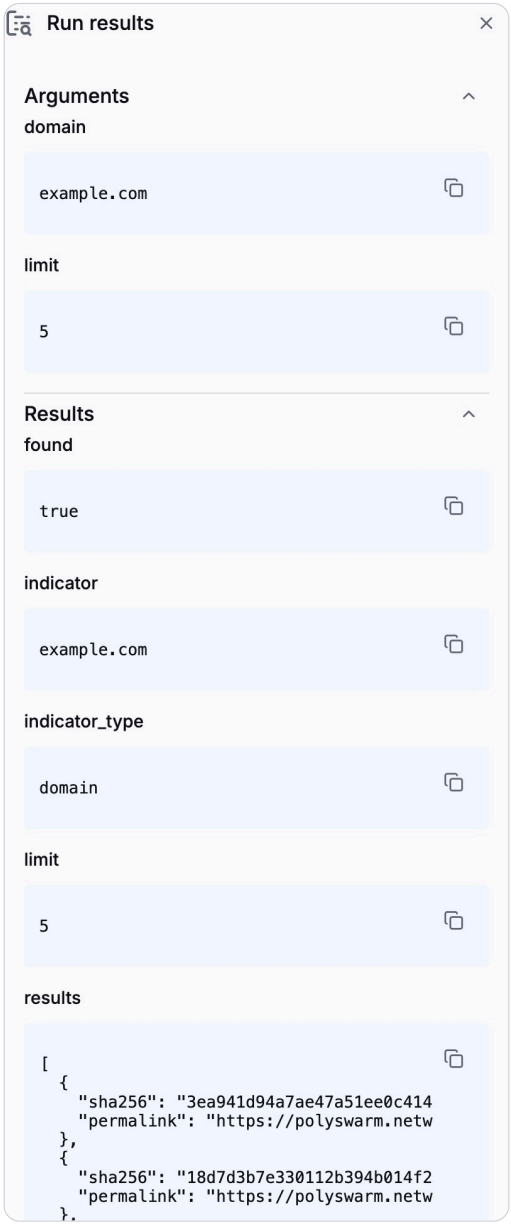

Figure 12: Search IOC returning artifacts associated with an indicator.

Hunt across PolySwarm's metadata

`Search Metadata` is the broadest read surface. It takes a query in Lucene style and returns matching artifact documents, so one query can combine a malware family, a tag, a score threshold and a time window. Set `query`, and use `limit`, `include` and `exclude` to shape the answer.

Field names matter, and the short form is the one that works. These three are confirmed against the live index:

- `polyunite.malware_family:OverlordRAT` for a family.
- `scan.latest_scan.polyscore:>0.99` for a score threshold.
- `artifact.created:>2026-11-01` for a time window.

Longer paths such as `metadata.polyunite.malware_family` return nothing rather than an error, so a query that silently finds zero results is usually a field name rather than an absent artifact.

Note that the query string you write is sent to PolySwarm, so treat it like any other field an analyst fills in.

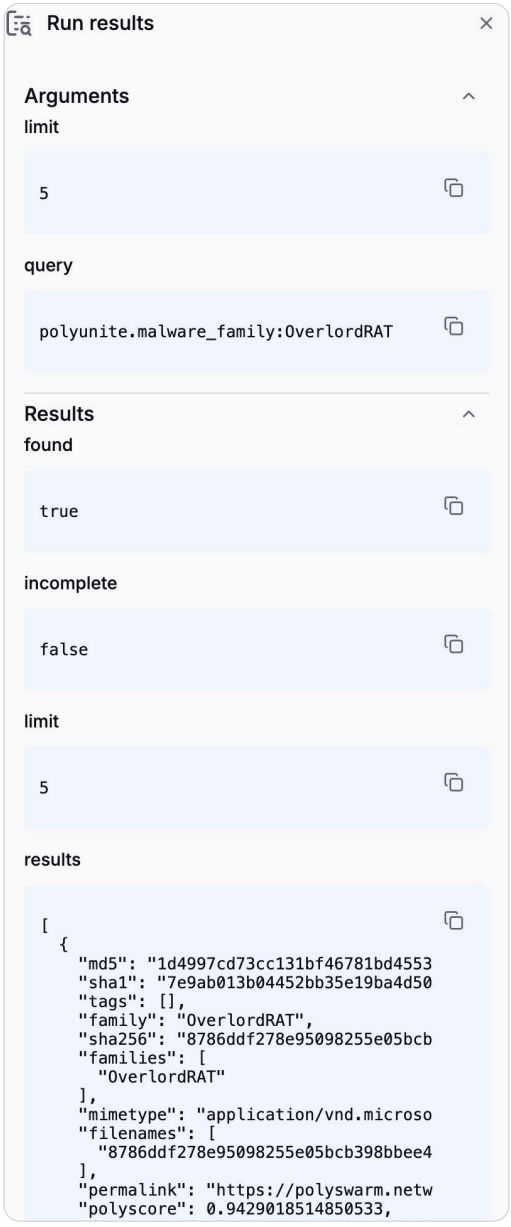

Figure 13: A metadata query and the artifacts it matched.

Get a fresh verdict on something unknown

When a lookup comes back unknown, the scan actions ask the engines for a new answer. `Scan File`, `Scan URL`, `Scan IP Address` and `Scan Domain` all submit the artifact and wait for the assertion window to close.

These spend against your plan, and what you submit is shared with every engine in your community. Decide deliberately before sending a customer file or an internal URL.

An existing result is reused unless `force_rescan` is set. One behaviour worth knowing: a finished result that no engine answered does not count as an existing result, so an artifact that no engine covers is submitted again on every run.

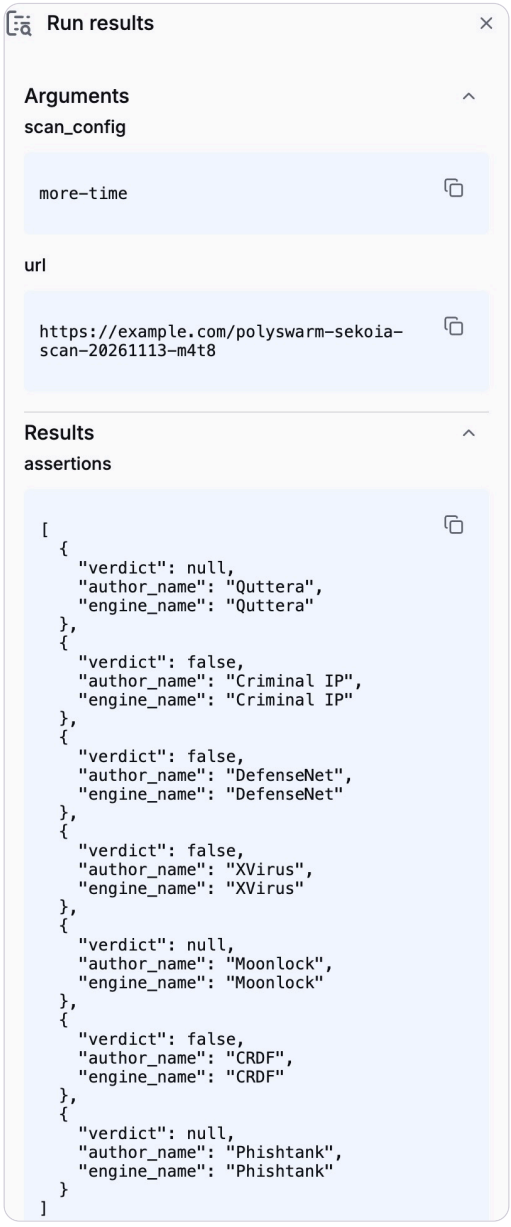

Figure 14: Scan URL returning a fresh multi-engine verdict.

Detonate something and read its behaviour

`Detonate File`, `Detonate URL` and `Detonate IP Address` run the artifact in a sandbox. `Get Sandbox Report` fetches the report afterwards, including the behaviour signatures, the MITRE technique identifiers, and the hosts, domains and URLs the sample contacted during the run.

A sandbox verdict and an engine verdict are different things and can disagree. A file every engine flags can behave inertly under detonation and score low. A low sandbox score means the sample did nothing in that sandbox, not that it is safe.

The screenshot below is exactly that case, which is why it is worth looking at closely. The sample is one seven engines call malicious, yet `malicious` reads false, because the detonation scored below the threshold for this provider. The behaviour block underneath still lists the hosts and domains it contacted and the signatures it fired. Read the behaviour, not just the flag.

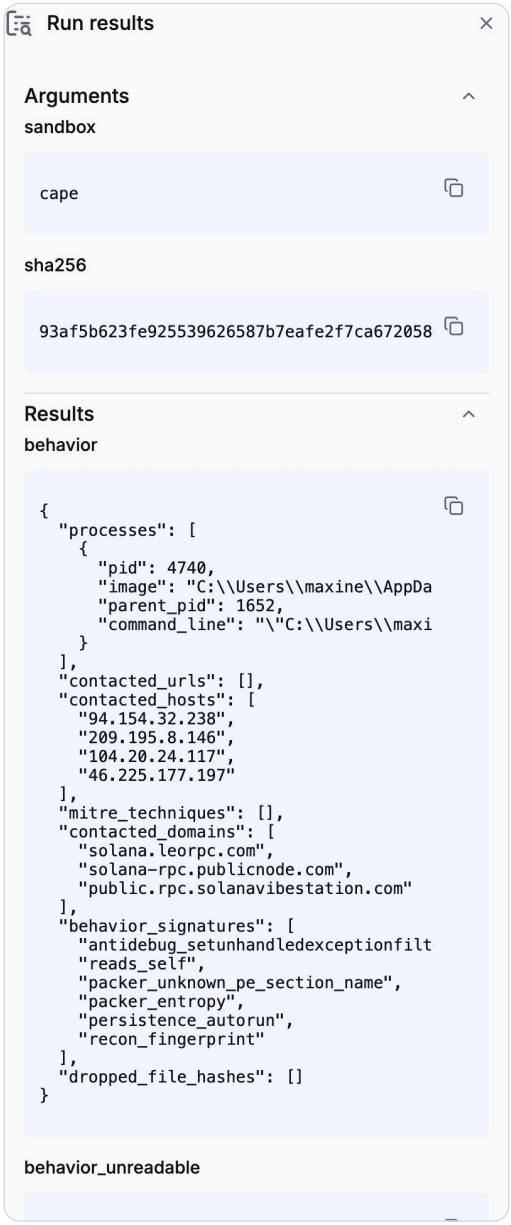

Figure 15: A sandbox report, showing behaviour and the hosts contacted.

React when a detonation finishes

A fresh detonation takes minutes, not seconds, and how many varies by artifact and sandbox: a URL detonation we timed finished in about four and a half minutes, while a large file can take considerably longer. Either way it is too long to block a playbook on. Submit the artifact with `wait` left off, then let the `Sandbox detonation completed` trigger start the playbook that handles the result.

The trigger polls for detonations your account created and emits one event per task that reaches a terminal state. The event carries the task identifier, the hash, the sandbox, the status, the community, whether a report is available, and both an API and a portal link to the result.

One behaviour to expect on day one: the first poll after you enable the trigger emits nothing. It records the detonations that already exist rather than replaying your history into the intake, and emits from the next new one onward.

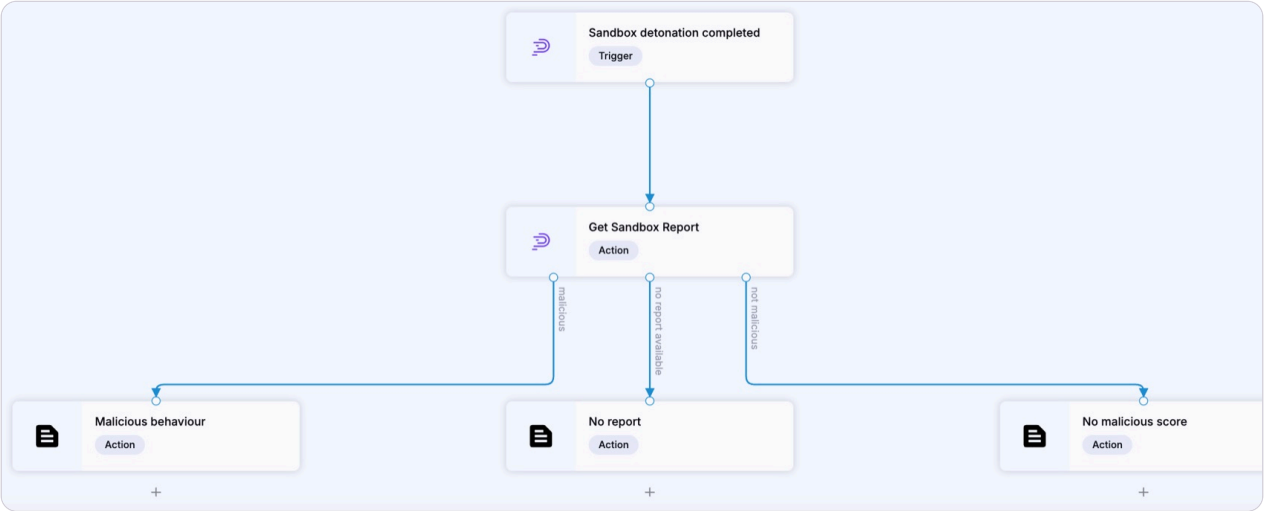

Figure 16: The sandbox completion trigger on a playbook canvas.

Pull indicators out of a block of text

`Extract IOCs` takes `text` or a `file` and returns the hashes, addresses, domains, URLs and email addresses inside it, including defanged forms such as `hxxp` and `[.]`.

This one is different from everything else here: it never calls PolySwarm. It runs entirely inside Sekoia, so the text stays in your environment and it costs nothing.



Figure 17: Extract IOCs turning a block of pasted text into indicators.

Check what your account can do

`Get Account Usage` reports the account behind the configured key, the plan, the communities it can read, and which features are entitled. Reach for it when actions start failing and you need to tell a bad key from a spent quota from a feature the plan does not include.

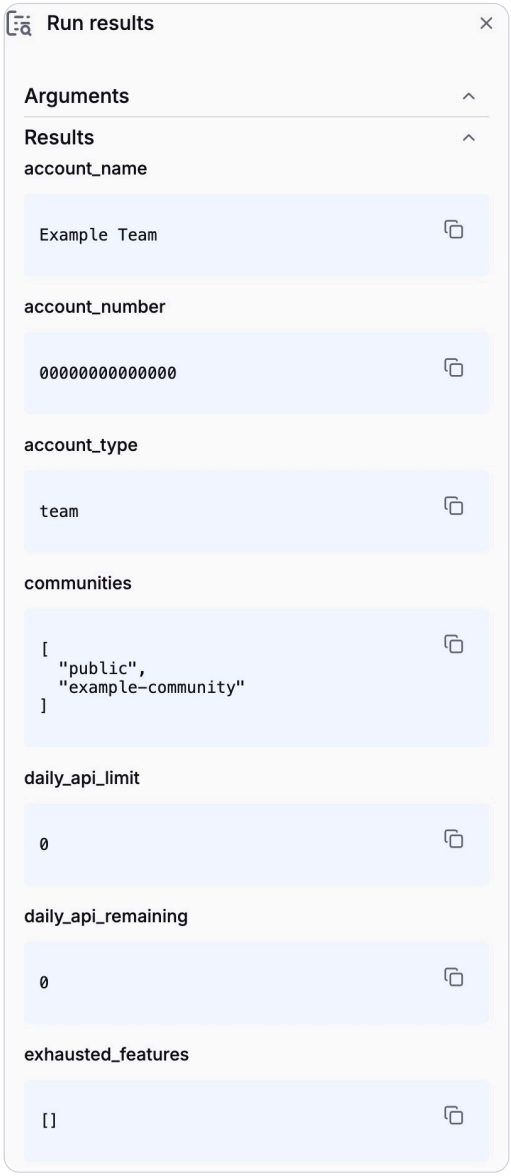

Figure 18: Get Account Usage, showing the account and its entitlements.

Where the playbook templates are

The six ready-made playbooks that use these actions are documented separately, in **PolySwarm Playbook Templates for Sekoia**. Start there if you would rather import a working playbook than wire one yourself.

We've submitted six playbook templates to Sekoia's Playbooks Catalog. Once Sekoia publishes them, you'll find them under `Start from a template`. Until then, ask customersuccess@polyswarm.io for the template files.

Quota and what things cost

The search actions are read only. They look up what PolySwarm already holds and spend no scanning quota, so they are the right default for anything that runs on every alert.

The scan actions submit the artifact to the engines and the detonate actions run it in a sandbox. Both spend against your plan.

Bulk hash lookup is a loop rather than a batch, so the cost is one request per hash, with duplicates charged once.

If you are unsure what your plan allows, run `Get Account Usage`. Note that the numeric daily limit and remaining count are only populated on plans that meter that way. On an unmetered plan both come back as zero, which does not mean you are out of quota: it means the plan does not count. The entitlement list is the useful part there.

What leaves your environment

A lookup sends the indicator. A scan or a detonation sends the artifact itself. An artifact you send is shared.

- **Nothing leaves.** `Extract IOCs` runs entirely inside Sekoia and never calls PolySwarm.
- **The API key only.** `Get Account Usage` asks what the account is entitled to. The completion trigger polls the status of detonations your account already created.
- **The indicator you looked up.** The search actions and `Get Sandbox Report` send the hash, URL, address or indicator you asked about and nothing else. These are read-only lookups and do not add the indicator to PolySwarm as a submission.
- **The artifact itself, and it is shared.** `Scan File` and `Detonate File` upload the file's contents. The URL, address and domain scan and detonate actions submit that value as an artifact. Whatever you submit is visible to every engine in the community your key belongs to and cannot be withdrawn afterwards. On the default public community that means every participating vendor, not only PolySwarm.

The scan and detonate actions refuse a private, loopback, reserved or otherwise non-public address before anything is sent, so a host from your own network is not submitted by accident. A URL that two parsers would read as different hosts is refused for the same reason.

Troubleshooting

Everything fails with an authentication error. The key is wrong, revoked, or belongs to a different community than the one configured. Run `Get Account Usage`: if that fails too, it is the key.

Lookups come back empty for artifacts you are certain exist. Almost always the `Community` field. A key scoped to a private community cannot read public results, and the reverse. Your PolySwarm contact can tell you the exact value to type.

An action reports a usage limit. The plan's quota is spent. Search actions cost no scanning quota, so they keep working when scans do not.

A scan times out. The assertion window had not closed within `max_wait_seconds`, which defaults to three hundred. The scan itself is not cancelled, so the result can be read later with a hash search.

A detonation seems to hang. It has not hung. A fresh run takes minutes, not seconds: one URL detonation we timed finished in about four and a half, and a large file takes longer. Use the completion trigger instead of waiting.

A result came back with no verdict. The window closed and no engine gave a malicious or benign answer. This is reported as unknown rather than as zero detections, on purpose. Engines abstain routinely, and coverage is thinnest on addresses and domains.

A bulk run reported `partial`. It stopped early and returned what it had. The hashes it never reached have no answer, and must not be read as clean.

Frequently asked

Does searching cost anything? No scanning quota. Search actions are read only.

Can I take back something I submitted? No. An artifact submitted to a community is visible to every engine in it and cannot be withdrawn. Decide before sending.

Why is unknown not the same as not detected? Because one means engines looked and found nothing, and the other means nobody answered. Treating the second as clean tells an analyst a sample is safe when nobody has examined it.

Why do the malicious and benign counts not add up to the total? Because an engine can abstain. The total counts engines that answered; malicious and benign count those that committed to a verdict. Abstentions are neither.

Which is right when the sandbox and the engines disagree? They measure different things. The engines are a static multi-vendor opinion; the sandbox is what the sample did in one environment. A low sandbox score means it did nothing there.

Do I need to deploy anything? No. The module runs on the Sekoia platform.

Support

- → Email customersuccess@polyswarm.io.
- → If lookups return nothing unexpectedly, check the Community setting on the connected account first.