Uninstall Guide

See also: the Installation Guide §7 (upgrading).

⚠ Read this first

Uninstalling a managed package destroys its custom objects and all their data. Salesforce deletes the object definitions shipped with the package; the records they held go with them. There is no recycle bin, no restore, no “undo”.

In concrete terms, in OrgGuardian's case, what disappears is: **all your findings**, **their entire handling history**, **all your metrics**, **all your signed compliance attestations**, **all your approved configuration baselines**, **all your no-code configuration**, and **all your trend history**.

If you are uninstalling to change version, read the Installation Guide §7 (upgrading) first: it states how a new version is installed. The version currently installed in your org is shown in *Setup → Installed Packages*.

Two prerequisites, without which the uninstall fails: remove all assignments of the OrgGuardian permission sets, including those made through the persona groups, and delete the configuration records that your no-code editors have written in the org (`reasonCode 10` refusal). The procedure is in §4.1, steps 5 and 6.

> Do the exercise at least once **before** you need it for real: uninstall on a sandbox copy, measure what you actually recover through the exports in §3, and then decide whether the content you accumulate in OrgGuardian deserves an external backup.

1. What is destroyed — the actual count

75 objects and types ship with the package: 29 `__c` objects (custom objects and custom settings), 7 Big Objects, 37 Custom Metadata Types and 2 platform events. They fall into four families that do not behave in the same way at uninstall. In your org, the package's detail page (*Setup → Installed Packages → OrgGuardian*) lists the components it installed.

The “Shipped label” column in the tables below carries each object's label as shipped, in English.

1.1 The 20 custom objects — your data goes with them

29 of these objects carry the `__c` suffix; **9** of them are **custom settings** (covered in §1.2). That leaves **20 custom objects** in the strict sense:

Object	Shipped label	What you lose
`Finding__c`	Finding	All findings, open and resolved alike
`Finding_Activity__c`	Finding Activity	Assignments, acknowledgements, snoozes, risk acceptances
`Org_Metric_Current__c`	Current Org Metric	All current values
`Scan_Run__c`	Scan Run	The marker of each past scan
`Insight__c`	Insight	The derived summaries
`Incident__c`	Incident	The groupings of correlated findings
`Attestation__c`	Attestation	**The compliance signatures** and their timestamps
`Config_Baseline__c`	Approved Configuration Baseline	**The signed configuration baselines**: the security state validated by your committee, frozen at a date, with its fingerprints, its signatory and its justification
`Config_Baseline_Item__c`	Approved Baseline Component	**The setting-level detail of the signed baselines**: one row per approved component, with its settings; read with its baseline, deleted with it
`Posture_Report__c`	Posture Report	The reports produced
`Org_Sla_Report__c`	Org SLA Report	The service-level reports
`Itsm_Ticket__c`	ITSM Ticket	The link between findings and the tickets in your tool
`Callout_Log__c`	Callout Log	The history of observed outbound calls
`Discovered_Endpoint__c`	Discovered Endpoint	The endpoint inventory built up
`Endpoint_Status__c`	Monitored Endpoint	The list of monitored endpoints and their status
`Config_Dependency__c`	Configuration Dependency	The computed dependency graph
`Linked_Org__c`	Linked Org	The multi-org settings
`Fleet_Finding__c`	Fleet Finding	The fleet inbox lines and their triage status; nothing is lost in the linked orgs, where nothing is written
`System_Log__c`	System Log	The package's internal log
`OrgGuardian_PurgeCursor__c`	OrgGuardian Purge Cursor	The progress state of the retention purge

1.2 The 9 custom settings — technical configuration, lost too

Custom setting	Shipped label	Content
`OrgGuardian_Install__c`	OrgGuardian Installation	The installation trace (version, date)
`OrgGuardian_Heroku__c`	OrgGuardian Heroku External Service	Endpoint, region, consent, key of the external collector
`OrgGuardian_Channel_Settings__c`	OrgGuardian Notification Channels	The mapping of channels to your Named Credentials
`OrgGuardian_Feature_Mute__c`	OrgGuardian Feature Mute	The mutes set by your administrators
`OrgGuardian_Demo__c`	OrgGuardian Demo	A demonstration-mode setting, with no effect in a customer org
`OrgGuardian_Plan_Trial__c`	OrgGuardian Plan Trial	The start date of the 30-day trial of the full plan, and the log retention window the nightly purge last applied
`OrgGuardian_Pseudonym__c`	OrgGuardian Subject Pseudonym	Your org's pseudonymization key; a reinstall creates a new one
`OrgGuardian_Alert_Preference__c`	OrgGuardian Alert Preference	Each user's alert delivery preferences
`OrgGuardian_Compliance_Map__c`	OrgGuardian Compliance Map	The version of the imported compliance mapping in force; empty, the packaged mapping applies. The imported file itself stays in your Salesforce Files

1.3 The 7 Big Objects — the hard part

They hold **the trend history**, that is, the one thing that cannot be rebuilt: the past.

Big Object	Shipped label
`Org_Limit_Snapshot__b`	Org Limit Snapshot
`Config_Snapshot__b`	Configuration Snapshot
`Compliance_Snapshot__b`	Compliance Snapshot
`Em_Metric_Snapshot__b`	Event Monitoring Snapshot
`Endpoint_Health_Snapshot__b`	Endpoint Health Snapshot
`Login_Baseline__b`	Login Baseline
`Prediction_Outcome__b`	Prediction Outcome

No export path in the product reads a Big Object in bulk. The product's screens do read some of them — to draw trends, or to **display** a past compliance snapshot — but that is screen-by-screen consultation, not a bulk export. Two exports draw on them: the configuration reconstruction of the **Drift & audit** tab rebuilds, from `Config_Snapshot__b`, the configuration state at a date you choose, or the difference between two dates (§3.1); the **Audit evidence** export carries, in one line, the head of the

chain of compliance snapshots (§3.3). Neither carries the history itself. Reinstalling therefore starts again from an **empty** history, which will rebuild itself at the pace of scans, without catching up on the past.

1.4 The 37 Custom Metadata Types — all your no-code configuration

Detection rules, thresholds, alert channels, scan windows, risk weighting, compliance frameworks, playbooks: all of this lives in records of Custom Metadata Types namespaced `orgguardian`, and goes with the package, **on one condition**: the records that your editors have written in the org must be deleted first, otherwise Salesforce refuses the uninstall (§4.1, step 6). **26** of these types are written by the product's no-code editors: the sections of **Administration ▸ Settings** (Feature activation, thresholds, channels, custom rules, posture digest, business criticality scopes, playbooks and the others), the rule pack import, and the fleet policies of the **Fleet** screen. The inventory of §4.1, step 6, finds the records concerned in your org.

The remaining **11** have no editor in the product: they contain only what the package ships, and losing them has no consequence for you. If you created records in one of them by hand, in Setup, the inventory of §4.1, step 6, finds them as well.

1.5 The platform events

`Org_Health_Alert__e` and `Callout_Log_Event__e` disappear with the package. Any automation **in your org** that subscribed to one of them stops working — and, if it references it through metadata, it **will block the uninstall** until you have removed it (§4.1).

---

2. What is NOT destroyed

- **Your business data.** OrgGuardian is read-only on it, and the uninstall deletes none of it.
- **Your own metadata.** Remote sites, Named Credentials, profiles, permission sets that *you* created: they belong to you, they stay. They simply become useless (see §5.2).
- **What has already left the org — and it is not all in the same place.** No outbound call is made by default: the package's 22 outbound features, in four families (alert channels 5, exports and integrations 4, external probes 7, self-callouts 6), are all closed at installation and open one by one. What those you opened have sent stays where it went, and two of them send to very different places:
    - **SIEM export — on your side.** It leaves through a *Named Credential* that **you** configured, towards **your** SIEM. The publisher does not receive it, does not store it and cannot erase it: it is your copy, under your retention policy. The uninstall does not touch it.
    - **Heroku option — on the publisher's side.** The telemetry goes to the address shown in the **Collector endpoint (https)** field (Administration ▸ Settings ▸ Heroku option, **Advanced settings**), which is pre-filled with the publisher's service and which your administrator authorises once as a remote site. What your org published there **remains after the uninstall**: uninstalling the package stops publication; it does not erase anything that has already left.

What this telemetry contains — and the answer splits in two, because the package and the service do not guarantee the same thing.

- **What the installed package PUBLISHES**: a posture summary of **twelve aggregate keys** — health score, total of open findings, five per-severity counters, number of metrics, region code, two timestamps, and your org's technical identifier. **No name, no email address, no IP address, no business record** is built by the sender: the payload is assembled from aggregates, masked, then checked by an egress guard before the call.
- **And a second content, if — and only if — you also enable certificate monitoring**: the **host names** of your external endpoints, read from your Named Credentials (a hundred at most), with your org's identifier and a timestamp. Three keys, nothing else — no login credentials, no secrets, no content from your calls. **This content does not go through masking**: the host names must arrive intact so that the service can perform the TLS handshake that your org cannot perform itself. The service does not store the host names and does not log them; it keeps only a daily count of the hosts probed, which the erasure below removes. If you do not want the host names to leave, leave this monitoring disabled, which is how it ships.
- **What the service RECEIVES and keeps**: the service **stores the received body as is**, without an allowlist of the twelve keys. What filters on arrival is a **refusal** (forbidden key vocabulary, inspected values, bounded values, body capped at 64 KB) that rejects a body resembling personal data — a barrier, not a list. The limit to twelve keys is therefore enforced by the package that you install and uninstall, not by the receiving service.

Retention is bounded: the service keeps only the **50 most recent summaries** per org.

How to have it erased. It does not disappear by itself, and you cannot delete it from your org: ask the publisher to erase it, quoting your org ID. The erasure exists on the publisher's side and it is final: a single operation removes, in one transaction, your registry row, your summaries, your usage counters and any attestation link open in your name, and reports the number of rows removed per table. It is irreversible: re-onboarding you later requires new keys.

> ⚠ **This guide sets no processing time**: contractual terms are those of your agreement with the publisher. To request the erasure, contact support at orgguardian.com/en/support.html, and ask for an acknowledgement of receipt.

A useful reflex before uninstalling, if you want the cut-off to be immediate: withdraw the Heroku option's consent (§4.1, step 2) **before** the uninstall, then request the erasure. In the other order, your last publication still goes out.

- **Your exports.** Any file you downloaded before the uninstall (§3).

---

3. Exporting beforehand — what is possible, and what is not

3.1 What the product can export

What it outputs	Format	Where in the product
The findings of the list, as filtered: severity, category, status (open by default), “My findings” and the list's other filters	CSV	**Findings** tab: **Export CSV**
The product's current configuration	JSON	**Executive** tab ▸ **Executive summary**, card **Config & posture export**: **Export config (JSON)** (administrators only)
The posture: non-PII findings + metrics + marker of the last scan	JSON	Same card: **Export posture (JSON)**
The same posture, flattened	CSV	Same card: **Export posture (CSV)**
The current compliance report	CSV	**Executive** tab ▸ **Compliance**: **Audit evidence (CSV)**
The same report	JSON	Same screen: **Audit evidence (JSON)** (only for users allowed to see amounts)
Your acceptance policies, custom detection rules and detector tunings (the “rule pack”), in a format the product imports again	JSON	**Administration** tab ▸ **Settings** ▸ **Custom rules**, card **Export / Import rules**: **Export rules**
The configuration state at a date you choose, or the difference between two dates, rebuilt from the configuration snapshots	CSV	**Drift & audit** tab, configuration reconstruction: **Export CSV**

Other screens also download what they show — fleet conformance, configuration impact, the access review of one user — as CSV, and the **Executive summary** downloads the current board pack as an HTML file (**Export board pack**, paid plan). They describe the current state, not the history that the uninstall destroys. One exception: the **Board report archives** card of the **Executive summary** lists the board reports stored in your org (`Posture_Report__c`, §1.1) and downloads each one as an HTML file, as stored on its date (**Show archives**, then **Download** on each row), on every plan. It is one file per report, with no bulk export; an archive that prints amounts needs a user allowed to see them.

These exports are triggered **from the screens**: the file is built in your org and saved by your browser. Building them involves **no outbound call** — no callout, no server-side PDF generation.

3.2 Two pitfalls to check before relying on them

Pitfall 1 — the configuration and posture exports sit behind a switch that is OFF by default. The **Config & posture export** switch (Administration ▸ Settings ▸ Feature activation) is **off** by default, and it requires a paid plan (technical tier `PRO` — the 30-day trial of the full plan counts). As long as the switch is off, or on the free plan, the card shows a locked panel instead of its three buttons: turn the switch on before the day of the uninstall, then open each file you download.

The same paid plan is needed by three other exports, each behind its own switch in the same **Feature activation** section. On the free plan, or with its switch off, the rule pack (switch on by default) is refused with a message, and the configuration reconstruction (switch **off** by default) reads nothing and says so. The compliance report (switch on by default) is the one that stays silent: the **Compliance** screen shows a locked panel, but its **Audit evidence (CSV)** button stays visible and returns a file reduced to its header line, with no error. **A silently empty export looks like a successful export.** Open the file and count the lines before uninstalling.

Pitfall 2 — a permissions refusal also degrades to a file with no findings, with no error. If the user cannot read the findings, the findings export returns a file **without a single finding row**, never an error. The file says so: its first line, which starts with `#`, announces that reading the findings was refused; the header line follows. The findings export also stops at **5,000** rows: when that cap is reached, the first line, again starting with `#`, announces a truncated export, and only the most recently seen findings are in the file. Same instruction: open the file and read its first line.

3.3 What the product CANNOT export

- **The 7 Big Objects** — no export path reads them in bulk (§1.3). This is the irreversible loss.
- **The signed attestations** (`Attestation__c`) and the chain of compliance snapshots (`Compliance_Snapshot__b`), whose chained fingerprints reveal any later change to a past compliance verdict: viewable on screen, not bulk-exportable from any product screen. The **Audit evidence** export carries only a digest of them: for each control, the state of its remediation plan read from the attestations (owner ID, due date, state, age, days overdue), and one line, starting with `#`, that gives the head of the chain (fingerprint, date of the latest snapshot, verification verdict). It carries neither the attestations themselves (signatory, timestamp, justification) nor the chain.
- **The approved configuration baselines** (`Config_Baseline__c`) and their setting-level detail (`Config_Baseline_Item__c`): viewable on screen, no dedicated export.
- **The finding handling history** (`Finding_Activity__c`): no dedicated export.
- **The Custom Metadata records** of your no-code configuration, except the rule pack: the configuration JSON (**Export config (JSON)**) gives a readable snapshot of part of it — feature switches, monitoring thresholds, alert channels, email alert templates, endpoint tag rules and maintenance windows — but it is **not** a re-importable format: there is no import function for it. Only the rule pack (§3.1) goes back in, through **Import rules** on the **Export / Import rules** card: your acceptance policies, custom detection rules and detector tunings.

3.4 What you are left with to fill these gaps

None of this is provided by the product; these are your platform tools:

- **Data Loader / Bulk API** on the package's `__c` objects, prefixing the namespace (`orgguardian__Finding__c`, `orgguardian__Attestation__c`, `orgguardian__Finding_Activity__c`…). This is the only complete route for the custom objects.
- **SOQL query on the Big Objects**: they accept only filters on their index, in index order. Test your query **before** D-day: a query that does not follow the index fails.
- **Metadata retrieve** (`sf project retrieve start --metadata CustomMetadata:…`) to freeze your configuration records as files.

- **The Salesforce uninstall screen** offers an option to temporarily keep a copy of the package's data. This is a **platform** behaviour, not an OrgGuardian feature: read what the screen states at the time of the uninstall, and do not treat it as an archive — the window is short and the scope covered is not guaranteed for Big Objects.

---

4. The procedure

4.1 Before

1. **Run your exports** (§3) and **open each file** to check that it is not empty.
2. **Close the outbound features you opened.** None is open at installation; if you opened any — the Heroku option, the SIEM export, an alert channel, a probe — turn its switch off first in the **Feature activation** section (Administration ▸ Settings ▸ Feature activation) and, for the Heroku option, click **Revoke the consent** (Administration ▸ Settings ▸ Heroku option, **Advanced settings**): that way you know that the package makes no more outbound calls, independently of the uninstall.
3. **Look for references from YOUR metadata to the package.** Report, dashboard, custom report type, flow, formula field, automation subscribed to `orgguardian__Org_Health_Alert__e` or `orgguardian__Callout_Log_Event__e`: Salesforce refuses an uninstall as long as a piece of metadata in your org references a component of the package. Remove these references beforehand, otherwise the uninstall will fail and you will have to start again.
4. **Write down your configuration.** The scan windows, thresholds, channels and rules that you adjusted will have to be re-entered in full, except what the rule pack carries (§3.1): keep its file, and **Import rules** restores your acceptance policies, custom detection rules and detector tunings after a reinstall (paid plan or trial, and the Salesforce permission “Modify Metadata Through Metadata API Functions”). The configuration JSON (**Export config (JSON)**) is a good aide-memoire for part of the rest — it is not an import.
5. **Remove the assignments of the OrgGuardian permission sets.** As long as a user holds one, Salesforce refuses the uninstall: *“This permission set is assigned to one or more users. You can only delete permission sets that aren't assigned to users.”* **First keep the list of who holds what**: none of the exports in §3 contains it, and it is what will let you reassign the personas after a reinstall (§5.3):

   ```
   sf data query --result-format csv --target-org <your-alias> --query \
     "SELECT Assignee.Username, PermissionSet.Name, \
     PermissionSetGroup.DeveloperName FROM PermissionSetAssignment \
     WHERE PermissionSet.IsOwnedByProfile = false \
     AND (PermissionSet.NamespacePrefix = 'orgguardian' \
     OR PermissionSetGroup.NamespacePrefix = 'orgguardian')" > assignments.csv
   ```

   Then remove them. By hand: Setup → **Permission Set Groups** → each `OrgGuardian_Persona_…` persona group → **Manage Assignments** (day-to-day access is assigned through these groups, the Permissions and Personas guide §3), then Setup → **Permission Sets** → each OrgGuardian

permission set → **Manage Assignments**. In one go, save the following anonymous Apex as `remove-assignments.apex`:

```
List<PermissionSetAssignment> assignments = [
    SELECT Id FROM PermissionSetAssignment
    WHERE PermissionSet.NamespacePrefix = 'orgguardian'
    AND PermissionSet.IsOwnedByProfile = false
];
delete assignments;
Integer remaining = [
    SELECT COUNT() FROM PermissionSetAssignment
    WHERE PermissionSet.NamespacePrefix = 'orgguardian'
    AND PermissionSet.IsOwnedByProfile = false
];
System.debug('REMAINING ' + remaining);
```

Then run it:

```
sf apex run --file remove-assignments.apex --target-org <your-alias>
```

The `REMAINING` line in the log must read **0**. Then check on screen that the four persona groups no longer have any assignment, and likewise any group that you may have composed yourself with an OrgGuardian permission set. This is lossless **if you kept `assignments.csv`**: reassigning is merely an operation to carry out, and the uninstall deletes these permission sets anyway.

6. **Delete the configuration that your no-code editors wrote in the org.** Every setting saved from the settings screens — every fleet policy saved from the Fleet screen, every rule pack imported — is a Custom Metadata record **created in your org**, in a package type: for Salesforce, it is metadata of yours that references the package, and the uninstall fails as long as any remain (`reasonCode 10` refusal). No product screen erases them, and Apex cannot either: Apex can write a Custom Metadata record, not delete one. Only a destructive deployment through the Metadata API can do it.

a. Inventory. The records shipped with the package and those written in your org live in the **same** package types; only their management state tells them apart. `installed`: shipped, it goes with the package. `unmanaged`: written in your org, it is the one that blocks.

```
rm -f inventory.json to-delete.txt
sf org list metadata --metadata-type CustomMetadata \
  --target-org <your-alias> --json > inventory.json
python3 - <<'EOF'
import json
import sys

PREFIX = "orgguardian__"
STOP = "INVENTORY UNREADABLE, draw no conclusion: "

def fail(reason): sys.exit(STOP + str(reason))
def full_name(row): return row.get("fullName") or ""
def is_ours(row): return full_name(row).startswith(PREFIX) or (
    row.get("namespacePrefix") == "orgguardian")
def prefixed(name): return name if name.startswith(PREFIX) else PREFIX + name

try: data = json.load(open("inventory.json", encoding="utf-8"))
except (OSError, ValueError) as error: fail(error)
if not isinstance(data, dict): fail("not a JSON object")
if data.get("status") != 0: fail(data.get("message"))
rows = data.get("result")
if isinstance(rows, dict): rows = [rows]
if not isinstance(rows, list): fail("no result list")

unmanaged = [row for row in rows if row.get("manageableState") == "unmanaged"]
names = sorted({prefixed(full_name(row)) for row in unmanaged if is_ours(row)})
with open("to-delete.txt", "w", encoding="utf-8") as target: target.write(
    "".join(name + "\n" for name in names))
print(len(names))
EOF
```

The last command displays a **number** and writes the corresponding names to `to-delete.txt`. If the inventory could not be read (unknown alias, expired session), it stops on `INVENTORY UNREADABLE` **without displaying a number or writing a file**: no number, no conclusion. It accepts names returned with or without the `orgguardian__` prefix and always writes them prefixed, the form that the destructive deployment requires in a subscriber org.

If the number is **0**, there is nothing to delete: go straight to §4.2.

b. Destructive deployment. In a folder of its own, a `package.xml` with no type and a `destructiveChangesPost.xml` that lists these names:

```
mkdir -p destructive
cat > destructive/package.xml <<'EOF'
<?xml version="1.0" encoding="UTF-8"?>
<Package xmlns="http://soap.sforce.com/2006/04/metadata">
    <version>67.0</version>
</Package>
EOF
cat > destructive/destructiveChangesPost.xml <<'EOF'
<?xml version="1.0" encoding="UTF-8"?>
<Package xmlns="http://soap.sforce.com/2006/04/metadata">
    <types>
EOF
sed 's|^|        <members>|; s|$|</members>|' to-delete.txt \
  >> destructive/destructiveChangesPost.xml
cat >> destructive/destructiveChangesPost.xml <<'EOF'
        <name>CustomMetadata</name>
    </types>
    <version>67.0</version>
</Package>
EOF
sf project deploy start --metadata-dir destructive \
  --target-org <your-alias> --wait 20
```

Read the **status** returned (`Succeeded`), not just the absence of an error on screen: this deployment is atomic; a single refused name and nothing is deleted.

c. Verification. Run inventory (a) again: it must display **0**. Only then uninstall. If the uninstall still refuses, read the component it cites: a forgotten Custom Metadata record is dealt with by going through a, b and c again.

4.2 The uninstall itself

As a precaution, first stop OrgGuardian's scheduled jobs (Setup → **Scheduled Jobs** → delete each job whose name starts with `OrgGuardian`). The shipped uninstall script tries to cancel them along the way (see below). Measured on 2026-10-04 on one of the publisher's orgs, with an earlier version of the package and eleven scheduled jobs in place: the platform did not refuse the uninstall, and no job remained after it, but the measurement does not show whether the script or the platform removed them. Stopping them first costs nothing, and it does not depend on the script being able to read `CronTrigger` (see below, and §4.3).

Setup → **Installed Packages** → **OrgGuardian** row → **Uninstall**.

Or from the command line, with the package version ID (`04t…`) that the command of §4.3 displays:

```
sf package uninstall --package <package-version-id> \
  --target-org <your-alias> --wait 30
```

The platform removes the package's components and their data. An uninstall script shipped with the package runs along the way: it **tries to cancel OrgGuardian's scheduled jobs**, those whose name starts with `OrgGuardian`, and it reads and deletes **nothing else**. It never raises an error: when it cannot read or cancel a job, it says so only in the debug log of the uninstall.

If the uninstall is refused on the dashboard folder. Message observed on 2026-10-16 on a Developer Edition: "OrgGuardian could not be removed during uninstall. Entity Type: Folder. Error: OrgGuardian: Before deleting this folder, you need to delete these dashboards from the folder and the Recycle Bin.: OrgGuardian Compliance, OrgGuardian Operations, OrgGuardian Posture Overview". These are the three dashboards shipped by the package. The platform deletes the "OrgGuardian" folder only when it is empty, Recycle Bin included, and the uninstall does not empty the Recycle Bin for you. Do what the message asks, then run the uninstall again: **Dashboards** tab → **OrgGuardian** folder → delete the named dashboards if they are still there; then **Recycle Bin** → select them → **Delete permanently**. No script of the package can do it before you: Salesforce runs no package code before an uninstall.

4.3 After — verify

```
sf package installed list --target-org <your-alias>
```

`orgguardian` must no longer appear in it. Then Setup → **Scheduled Jobs**: no job whose name starts with `OrgGuardian` should remain. If one remains, cancel it by hand: the absence of an error during the uninstall does not prove that the script could cancel every job (the Troubleshooting Guide §10).

---

5. After the uninstall

5.1 What does not come back

A reinstall gives you back the **product**, not your **past**. You start again from a clean org: zero `Scan_Run__c`, zero `Finding__c`, zero metrics, zero snapshots, no attestations, and the package's default configuration.

And every feature that ships off comes back **off**: the package embeds **no `Feature_Gate__mdt` record** that could turn a feature switch on at installation. None of the metadata records it ships is a switch: they are texts (`Org_Insight`, `Posture_Narrative`, `Posture_Recommendation`, `Remediation_Playbook`, `Root_Cause_Rule`) and reference rules, tables or estimates (`Attack_Path_Rule`, `Data_Jurisdiction`, `Parity_Rule`, `Release_Calendar`, `Remediation_Effort`). You can check it right after reinstalling, before any activation: `SELECT COUNT() FROM orgguardian__Feature_Gate__mdt` returns **0**. Any activation you had made must be redone, one by one (the Configuration Guide); conversely, a feature that is on by default and that you had turned off comes back **on**.

5.2 What remains, and becomes orphaned

- **The remote sites** that you had declared for the Heroku option (Setup → Security → Remote Site Settings): they remain, they no longer serve any purpose. Delete them if you are not reinstalling — an egress authorisation with no use is a free attack surface.
- **The Named Credentials** created for the alert channels, the SIEM export, the ITSM integration, multi-org or the probes: likewise.
- **The external client app** created for **Tooling API access** (Administration ▸ Settings ▸ Tooling API access), and its run-as user: the package's own credentials, which held its consumer key and secret, go with the package, but the app stays in your org with its client credentials flow still enabled. Delete it, or disable that flow, and deactivate the run-as user if it served nothing else.
- **Permission set assignments** do not go away on their own: they block the uninstall and must be removed beforehand (§4.1, step 5). After a successful uninstall, none therefore remain.
- **Permission set groups** that you may have composed yourself by including an OrgGuardian permission set: such a group is metadata of your org that references the package, so it falls under step 3 of §4.1 (remove the OrgGuardian permission set from the group before uninstalling). After reinstalling, recompose the group.

5.3 If you were uninstalling to change version

Continue with the Installation Guide §3, then §5 in full: assignment of the personas, bootstrap, re-enabling of the features, and re-entry of the configuration, after the import of your rule pack if you exported one (§3.1). Allow time for the re-entry, not just for the installation.