TA-trackme-lookupmonitor

A Splunk Technology Add-on that exposes the `| trackmelookupsmonitor` generative custom search command — a fast, low-overhead way to enumerate Splunk lookups (CSV files and KVstore collections) and report their last-modification time and record count.

Built and consumed by TrackMe for Splunk as the data source behind the **Lookups monitoring** search mode of the Hybrid Trackers wizard, but the command is also useful standalone for inventory, debugging and ad-hoc audits.

- **Download**: https://downloads.trackme-solutions.com/TA-trackme-lookupmonitor/
- **Repository**: https://github.com/trackme-limited/TA-trackme-lookupmonitor
- **License**: TrackMe EULA — https://docs.trackme-solutions.com/latest/license.html

---

What the command does

For every lookup matching the configured filter, the command emits one event with:

- `_time` = the lookup's modification time (so events render properly on the Splunk timeline).
- `_raw` = a JSON dump of the row (so the events view shows the structured payload as the body).
- The CSV file's **filesystem** modification time (`os.stat(path).st_mtime`) — not the unreliable knowledge-object mtime that `/services/data/lookup-table-files` exposes.
- The CSV **row count**, obtained with a `csv.reader`-based streaming counter that respects RFC 4180 quoted-newline cells and handles `.csv.gz` natively.
- For KVstore collections, the **document count** (cursor-paginated over `_key`, capped at 5 M to avoid runaway counts) and a parsed modification time read from an operator-supplied list of candidate fields (see below).

The command runs locally on the search head where the search is dispatched (`local = true` in `commands.conf`). For remote search heads, install the TA there too and wrap the command with `splunkremotesearch` from TrackMe.

Quick start

Enumerate every CSV lookup in the `search` app:

```
| trackmelookupsmonitor app_namespace="search" lookup_type="csv"
```

Discovery across the `search` app and every app whose name starts with `TA-` or `Splunk_`:

```
| trackmelookupsmonitor app_namespace="search, TA-*, Splunk_*" lookup_type="csv"
```

Track KVstore collections that store their modification time under `updated_at` (with `_time` and `mtime` as fallbacks):

```
| trackmelookupsmonitor lookup_type="kvstore" \
    kvstore_time_fields="updated_at, _time, mtime"
```

Both CSV and KVstore, no filter, no tenant context — full lookup inventory for the running user:

```
| trackmelookupsmonitor lookup_type="both"
```

Options

Option	Required	Default	Description
`tenant_id`	no	`""`	TrackMe tenant identifier. Stamped on every row. Omit for standalone use.

Option	Required	Default	Description
`app_namespace`	no	`-`	Single app name, glob (e.g. `TA-*`), or **comma-separated list mixing both** (`search, TA-*, Splunk_*`). `-` or `*` matches every app. Whitespace around list items is ignored; an empty item (e.g. trailing comma) is skipped; if any item is `-` or `*` the whole spec collapses to match-all.
`name_pattern`	no	`.*`	Regex (`re.search`) applied to the lookup name. Backslash escapes like `\d`, `\w`, `\.` are preserved.
`lookup_type`	no	`csv`	One of `csv`, `kvstore`, `both`.
`kvstore_time_fields`	no	`_time, mtime, updated_at, modified, timestamp, last_modified`	Comma-separated, preference-ordered list of candidate field names that may carry the per-record modification time of KVstore documents. Splunkd only auto-maintains `_key` and `_user` — there is **no** automatic timestamp field — so this list is how operators tell the command which field(s) their collections use. The command tries each candidate in order and the first one with a parseable value (epoch number, numeric string, or ISO 8601) wins; `mtime_source` on the emitted row reports which field was read. When no candidate resolves, the command logs a diagnostic line listing the candidates tried alongside the collection's actual top-level field names, so extending the list is a one-round-trip fix.

Option	Required	Default	Description
`kvstore_time_fields_ignore_case`	no	`false`	Resolve `kvstore_time_fields` candidates **case-insensitively** against the collection's actual field names. KVstore is backed by MongoDB where field names are case-sensitive, so by default `Last_Seen` does not match `last_seen`. Turn this on when monitoring collections that use heterogeneous naming conventions: the command samples one document per collection, builds a case-folded name map, and logs at INFO when a candidate is flipped (e.g. `'Last_Seen' resolved case-insensitively to 'last_seen'`). Adds at most one extra REST call per collection.
`kvstore_page_size`	no	`10000`	Number of KVstore documents fetched per REST page during the cursor-paginated `_key` walk that powers the document count. Larger pages mean fewer HTTP round-trips but more memory per page on both splunkd and the command.
`kvstore_count_cap`	no	`0`	Hard ceiling on the counted document total per KVstore collection. `0` means **no cap** — walk the entire collection regardless of size. When the cap is reached the emitted row carries `kvstore_count_truncated=1` so the count is unambiguously bounded rather than exact. Useful for very large collections where an exact count is not needed and the cursor walk would dominate the tracker's run time.
`max_workers`	no	`min(cpu*2, 8)`	Thread-pool size used to inspect lookups in parallel. CSV file IO benefits from parallelism; KVstore REST calls are internally serialised behind a lock to respect splunklib's single-connection model.
`register_component`	no	`false`	Reserved for the TrackMe integration.
`component`	no	`null`	Reserved for the TrackMe integration.

Option	Required	Default	Description
`report`	no	`null`	Reserved for the TrackMe integration.

Output fields

Field	Meaning
`_time`	Mtime (epoch seconds) — falls back to `now()` when mtime cannot be resolved.
`_raw`	JSON dump of the row (so the Splunk events view shows the structured payload as the event body).
`object`	Stable identifier `lookups:<app>:<name>` — used as the entity key by TrackMe.
`data_index`	Always `lookups` (the pseudo-index that the TrackMe Hybrid Tracker lookups mode uses to classify entities).
`data_sourcetype`	`<app>:<name>` — unique per discovered lookup.
`data_last_time_seen`, `data_last_ingest`, `data_first_time_seen`	The resolved modification time. For CSV: the filesystem mtime. For KVstore: the value read from the first candidate field in `kvstore_time_fields` that yielded a parseable value.
`data_eventcount`, `latest_eventcount_5m`	Record count. `null` if the count step failed (look at `status_message` for the reason).
`lookup_type`	`csv`, `csv_gz`, `kvstore`, or `other` (non-CSV file payloads like `.kmz`, `.geo`).
`lookup_path`	Absolute filesystem path for CSV; `<app>:<collection>` for KVstore.
`lookup_size_bytes`	(CSV only) file size in bytes from `os.stat`.
`app_namespace`	The Splunk app the lookup lives in.
`mtime_source`	`fs` (CSV filesystem mtime), `kvstore:<field>` naming the candidate field that yielded the KVstore mtime, or `unavailable` when no candidate field had a parseable value.
`kvstore_count_truncated`	`1` if the cursor walk hit the `kvstore_count_cap` ceiling before exhausting the collection, `0` otherwise. Always `0` when `kvstore_count_cap=0` (the default), which walks the entire collection.
`status_message`	Optional — present on failure rows (`stat failed`, `row count failed`, `kvstore count failed`, `invalid name_pattern regex`).
`splunk_server`	Host that ran the inspection.
`tenant_id`	Echoed from the input option (or `""` when omitted).

How modification times are resolved

CSV lookups

The command calls `/services/data/lookup-table-files` to enumerate visible lookups and resolve each one's on-disk path (`content.eai:data`), then `os.stat(path)` for the filesystem mtime. That is **deliberately** not the `updated` mtime that the REST endpoint exposes, which reflects the *knowledge-object* mtime (touched whenever permissions or props change) and is not a reliable proxy for "when did the file last change?".

`mtime_source = "fs"` on the emitted row signals this.

KVstore collections

KVstore is the difficult case: splunkd only auto-maintains `_key` and `_user` per record. There is **no** automatic `_time` field — it is only present if the storing application explicitly sets it. Common conventions vary widely (`_time`, `mtime`, `updated_at`, `modified`, `timestamp`, ...).

The command takes a comma-separated, preference-ordered list of candidate fields via the `kvstore_time_fields` option (default covers the most common conventions, see Options above), then for each candidate runs:

```
GET /servicesNS/-/<app>/storage/collections/data/<col>
    ?fields=<field>&sort=<field>:-1&limit=1
```

The first candidate that yields a parseable value wins. Values are coerced via a permissive parser that accepts epoch numbers, numeric strings, and ISO 8601 strings (with `Z`, `+HH:MM`, or `+HHMM` suffixes, and naive timestamps treated as UTC).

The emitted row carries `mtime_source = "kvstore:<field>"` so operators can see which field was actually read. When none of the candidates work, `mtime_source = "unavailable"` and `data_last_time_seen` is null — and the command emits a single diagnostic `WARNING` to `trackmelookupsmonitor.log` listing the candidates tried alongside the collection's actual top-level field names, so extending `kvstore_time_fields` is a one-round-trip fix.

Case-sensitive field names KVstore is backed by MongoDB and field names are case-sensitive — `Last_Seen` and `last_seen` are two different fields. By default the command matches candidates exactly, which is the right behaviour when your collections are consistent. When monitoring collections that use heterogeneous naming conventions (e.g. one collection uses `Last_Seen`, another uses `last_seen`), set `kvstore_time_fields_ignore_case=true`. The command then samples one document per collection, builds a case-folded name map, and resolves each candidate to the collection's actual field name; the resolution is logged at INFO (e.g. `KVstore candidate 'Last_Seen' resolved case-insensitively to 'last_seen' for myapp:mycollection`) and the chosen field still appears in `mtime_source = "kvstore:<actual-field-name>"` on the emitted row.

Document count The document count is obtained by cursor-walking the collection on `_key` (which is always indexed). Two options tune this:

- `kvstore_page_size` (default `10000`) — documents per REST page. Larger pages reduce HTTP round-trips at the cost of more memory per page.
- `kvstore_count_cap` (default `0`) — hard ceiling on the counted total; `0` means "no cap, walk the entire collection". When the cap is reached the emitted row carries `kvstore_count_truncated = 1`.

Logging

The command logs at INFO level to `$SPLUNK_HOME/var/log/splunk/trackmelookupsmonitor.log` (rotated, 10 MB × 1 backup). The TA is hidden from the launcher (`is_visible = false`) because operators interact with the command directly or via TrackMe.

Remote search heads

The command is forced-local (`local = true`) so it always runs on the search head that dispatches the search. To monitor lookups on a remote SH, install the TA there too and wrap the command call in TrackMe's `splunkremotesearch`:

```
| splunkremotesearch account="<remote>" search="| trackmelookupsmonitor lookup_type=\"csv\""
```

The remote SH only needs **this** TA — TrackMe itself does not have to be deployed there. That is one of the reasons the command lives in a dedicated TA rather than inside the main TrackMe app.

Building from source

The build uses the Splunk UCC framework. Build dependencies are listed in `requirements.txt`; runtime dependencies live in `package/lib/requirements.txt` and are vendored into `output/<appid>/lib/` by `ucc-gen` on every build: `splunk-sdk` (the only library the command imports) and `splunktaucclib`, which ucc-gen 6.x requires whenever a `globalConfig.json` exists and whose gRPC / protobuf / OpenTelemetry chain is stripped via `package/lib/exclude.txt`. `build/check_runtime.py` then verifies the built package under Python 3.9 and 3.13 in CI.

```
pip install -r requirements.txt
cd build && python3 build.py --keep
```

Produces `output/TA-trackme-lookupmonitor_v<ver>_<epoch>.tgz` and side-files (`version.txt`, `version_full.txt`, `build.txt`, `release-sha256.txt`).

To also submit the build to Splunk AppInspect:

```
cd build && python3 build.py --keep --submitappinspect \
    --userappinspect "$SPLUNK_BASE_LOGIN" \
    --passappinspect "$SPLUNK_BASE_PASSWD"
```

The CI (`.github/workflows/github-actions-build.yml`) runs this automatically on every push and uploads the AppInspect HTML/JSON reports as workflow artefacts. On `main`, the workflow also pushes the build to `downloads.trackme-solutions.com`.

Compatibility

- **Splunk**: 9.x and above (Splunk Cloud certified via the same AppInspect tags as the rest of TrackMe).
- **Python**: 3.9 to 3.13 (declared in `commands.conf` via `python.required = 3.9,3.13`).
- **Deployments**: standalone, distributed, and search-head clustering.

Changelog

See GitHub Releases for the per-version notes. Highlights:

- **1.0.9** — Dependency refresh: bundled `splunk-sdk` (splunklib) upgraded to 3.0.0 (core SDK runs on both Python 3.9 and 3.13 as declared in `commands.conf`; only the unused `splunklib.ai` subpackage needs 3.13) and the build moved to UCC 6.6.0 on Python 3.13. `splunktaucclib` 8.2.0 stays vendored (ucc-gen 6.x requires it), but the gRPC / protobuf / OpenTelemetry chain that its `solnlib` dependency now pulls in (unused by the TA, ~13 MB of dead weight including a compiled gRPC extension) is stripped at build time via `package/lib/exclude.txt`, bringing the package back to ~0.5 MB. No functional change to the command.
- **1.0.8** — Fix the duplicate `TRUNCATE` option that blocked AppInspect Cloud vetting. Remove the unused UCC logging settings page and its generated REST/Mako surface, resolving the Python handler declaration and custom Mako template findings for future Splunk releases.
- **1.0.7** — Skip disabled CSV and KVstore knowledge objects during discovery. KVstore mtime probe gains `kvstore_time_fields_ignore_case` (case-insensitive resolution against actual collection fields, off by default). KVstore document count gains `kvstore_page_size` (default `10000`, was hard-coded) and `kvstore_count_cap` (default `0` = no cap, was hard-coded at 5 M). Diagnostic logging on probe failure now lists the candidates tried alongside the collection's actual top-level field names.
- **1.0.6** — KVstore data reads no longer pass `owner="-"` (splunkd rejects the wildcard); inherits the dispatching user's context.
- **1.0.5** — KVstore mtime probe accepts an operator-supplied `kvstore_time_fields` candidate list (replaces the previous `_time`-only assumption).
- **1.0.4** — `tenant_id` is now optional, so the command can be invoked standalone.
- **1.0.3** — `app_namespace` accepts a comma-separated list of names / globs.
- **1.0.2** — Every emitted row carries `_time` and `_raw` so events render on the Splunk timeline.
- **1.0.1** — Switched the build to UCC (vendors `splunklib` automatically); added logging settings page.
- **1.0.0** — Initial release.