

CONTROL TOWER FOR CLAUDE · GUIDE

NIST AI RMF and Claude: what you can show, and a script to check it

A script, run on generated records, that checks an AI incident log against the nine example fields NIST lists, a Claude use-case register for a context of use, and the Activity Feed for days with no records.

Published 2026-10-31 · Online version and downloads: www.datameaning.com/guides/nist-ai-rmf-evidence-for-claude/

A company using Claude Enterprise can show three records: an AI incident log with the nine example fields NIST lists, a register of Claude use cases that each state a context of use, and Activity Feed records by day from the day recording began, with any day that has none listed. This guide's script checks those three records and lists the gaps.

What you will need

- The Compliance API turned on for your organization, the day you first turned it on, and a key that can read the Activity Feed: a Compliance Access Key or an Admin API key with the `read:compliance_activities` scope ([Anthropic: Query the Activity Feed](#) (read 2026-10-31)). The script reads a file and does not call the API: you pull the feed yourself, as one JSON object per line, from the day you first turned the Compliance API on.
- Your AI incident log as a CSV with a header row, one row per incident, with a column for each of NIST's nine fields. Keep it in your own system of record; it is not in the feed.
- Your Claude use-case register as a CSV: for each use case an id, its name, the department, the intended purpose, the context of use, an owner, a risk tier and the day it was last reviewed.
- Python 3 and the `duckdb` module.

The steps in full

Step 1. Pin NIST's text, and set what you can disagree with

The script downloads both NIST documents and prints each one's size, its Last-Modified header and its SHA-256. If either differs from the pinned size or hash, an input check fails, so you know NIST re-published the file and the quoted action text needs reading again. The pinned files are the July 2024 [NIST AI 600-1](#) (read 2026-10-31) and [NIST AI 100-1](#) (read 2026-10-31), the AI RMF 1.0. If the download fails, the check fails too: a check that could not run is not a pass.

```
def pin_nist():
    """Download each NIST document once, print its size, Last-Modified and SHA-256, return the
    number that do not match."""
    bad = 0
    for label, url, size, sha in NIST_DOCS:
        try:
            req = urllib.request.Request(url, headers={"User-Agent": USER_AGENT})
            with urllib.request.urlopen(req, timeout=60) as r:
                body, modified = r.read(), r.headers.get("Last-Modified", "not sent")
            except (OSError, http.client.HTTPException) as e:
                # a cut-off download is a failed
pin, not a crash
                print(f" {label}: could not be fetched ({e.__class__.__name__}); the pin cannot be
checked")
                bad += 1
                continue
            got = hashlib.sha256(body).hexdigest()
            same = len(body) == size and got == sha
            bad += 0 if same else 1
            print(f" {label}: {len(body)} bytes, Last-Modified {modified}, sha256 {got}" + (" " if same
else " ** DIFFERENT FROM THE PIN **"))
        return bad
```

Everything you can disagree with sits in one block at the top. The period, the review interval (180 days is our setting, not NIST's: NIST names none), the column of your files that holds each field, and the words that mark an HR use case. NIST's own text reads "Data Reported". The script reads that field as the date the incident was reported and checks it as a date; change the column or the check if you read it differently. NIST says the System ID is "auto-generated most likely"; the script accepts any text you use as the id.

```
# ---- CHANGE THESE FOR YOUR OWN DATA -----
PERIOD_START = "2026-07-06"           # first day of the period you are evidencing (UTC, inclusive)
PERIOD_END = "2026-09-27"             # last day of the period (UTC, inclusive)
COMPLIANCE_API_ENABLED_ON = None       # the day you first turned the Compliance API on, YYYY-MM-DD;
None fails a check
KNOWN_GAP_DAYS = []                   # days BEFORE the period with no records that you explained
earlier, YYYY-MM-DD
ACTIVITIES_FILE = None                 # your Activity Feed pull, one JSON object per line; None =
generate
INCIDENTS_FILE = None                  # your AI incident log (CSV with a header row); None =
generate
USE_CASES_FILE = None                  # your Claude use-case register (CSV with a header row); None
```

```

= generate
REVIEW_AS_OF = "2026-09-27"          # the day review ages are counted from
REVIEW_MAX_DAYS = 180                # your own policy: a register row not reviewed within this
many days is overdue
# the column of YOUR incident log that holds each of NIST's nine fields (the defaults are NIST's
own words)
INC_COLS = {
    "system_id": "System ID", "title": "Title", "reporter": "Reporter", "system_source":
"System/Source",
    "date_reported": "Data Reported", "date_of_incident": "Date of Incident", "description":
"Description",
    "impacts": "Impact(s)", "stakeholders": "Stakeholder(s) Impacted",
}
# the column of YOUR register that holds each of its fields
UC_COLS = {
    "use_case_id": "use_case_id", "use_case": "use_case", "department": "department",
    "intended_purpose": "intended_purpose", "context_of_use": "context_of_use", "owner": "owner",
    "risk_tier": "risk_tier", "last_reviewed": "last_reviewed",
}
# HR and people use cases are listed for Legal; change the words to fit your register
HR_TERMS = ["hiring", "recruit", "resume", "candidate", "promotion", "performance review",
"termination",
    "employee monitoring"] # matched inside use_case and
intended_purpose, any case
HR_DEPARTMENTS = ["hr", "human resources", "people", "talent"] # matched against the whole
department value, any case
LEGAL_REASON = ("laws on automated employment decisions, such as NYC Local Law 144, Colorado SB 26-
189 (bill text not "
    "verified) and California's automated decisionmaking technology rules under the
CCPA, may apply. "
    "Other laws may also apply")
# -----

```

The generated incident log uses NIST's own words as its column names, so a log written that way needs no column changes. Column names are matched without regard to case, because DuckDB identifiers are case-insensitive, quoted ones too ([DuckDB: keywords and identifiers](#) (read 2026-10-31)). If a name in the block does not exist in your file, the script stops and prints the columns it did find.

Step 2. Read dates strictly

Files are read as text with `all_varchar` and a header row ([DuckDB: CSV overview](#) (read 2026-10-31)). DuckDB's own cast to a date is lenient: it reads 8/5/26 as the year 8 and accepts 2026/09/24. So the script checks the form first and only then parses; `try_strptime` returns NULL when the text does not match the format ([DuckDB: date format functions](#) (read 2026-10-31)). A date in any other form becomes NULL, and the row is listed with `NOT WRITTEN YYYY-MM-DD`. No row is dropped for a date it could not read.

```

def strict_date(col):
    """YYYY-MM-DD only. DuckDB's own cast is lenient: it reads 8/5/26 as the year 8."""
    c = f"trim(coalesce({qcol(col)}, ''))"

```

```
return f"CASE WHEN regexp_full_match({c}, '{DATE_FORM}') THEN try_strptime({c}, '%Y-%m-%d')::DATE END"
```

Step 3. Run ten input checks before building anything

A list built from an incomplete input looks complete. The script runs ten checks first, and each prints the planted fault that trips it. If any fails, all three lists are printed with `INCOMPLETE` in their headings.

- **NIST documents that are not the pinned file** (a re-published file, or a failed download).
- **Duplicate activity ids** (a duplicated record).
- **Activity records with no id or no readable `created_at`** (an unreadable timestamp).
- **The pull does not reach back to the day the Compliance API was turned on** (a pull that starts at the period). If `COMPLIANCE_API_ENABLED_ON` is not set, it cannot be checked, and the check fails instead.
- **Days with no records, from enablement (or the period's start) to its end** (days removed from the feed). Days before the period that you explained earlier go in `KNOWN_GAP_DAYS` and print as information; a day inside the period always counts.
- **Days in the period before the Compliance API was enabled** (the Compliance API turned on late).
- **Incident log rows repeated**, with the same System ID, Title and Date of Incident (a duplicated incident row).
- **Register ids listed more than once** (a duplicated register row).
- **Register rows with no use case id** (a register row with no id).
- **Incident log and register dates not written `YYYY-MM-DD`** (a short date such as 8/5/26).
The row keeps its own status too, so it is never dropped.

The script flags duplicates and does not remove them. Ids in both files are lower-cased and trimmed before they are matched, so `uc-001` in the log and `UC-001` in the register are the same use case.

Step 4. The incident log against NIST's nine fields

NIST gives the nine fields as examples ("such as"). The script checks all nine because we chose to. Every row of the log is listed. None is filtered out by date, because an incident with an unreadable date is the one you most need to see. A blank field counts as missing after trimming. A row gets every status that applies, joined with a semicolon. The output shows the row's place in the file, the System ID and the Date of Incident, and never the reporter, the title or the description, so the list carries no person's name.

```
missing = ", ".join(f"CASE WHEN {k}{'_text' if k in INC_DATES else ''} {'=' + chr(39) * 2 if k in INC_DATES else 'IS NULL'} THEN '{INC_LABELS[k]}' END"
                    for k in INC_LABELS)
```

```

unread = ", ".join(f"CASE WHEN {k}_text <> '' AND {k} IS NULL THEN '{INC_LABELS[k]}' END" for k
in INC_DATES)
sql = f"""
    WITH reg AS (SELECT DISTINCT use_case_id FROM register WHERE use_case_id IS NOT NULL),
    i AS (SELECT n.*, concat_ws(' ', {missing}) AS missing, concat_ws(' ', {unread}) AS
unread,
        r.use_case_id IS NULL AS off_register
    FROM (SELECT *, lower(system_id) AS sid FROM incidents) n LEFT JOIN reg r ON
r.use_case_id = n.sid)
    SELECT row_no, coalesce(system_id, '-') AS system_id, date_of_incident,
        coalesce(nullif(concat_ws(' ',
            CASE WHEN missing <> '' THEN 'MISSING: ' || missing END,
            CASE WHEN unread <> '' THEN 'NOT WRITTEN YYYY-MM-DD: ' || unread END,
            CASE WHEN system_id IS NOT NULL AND off_register THEN 'SYSTEM NOT ON REGISTER'
        END,
        CASE WHEN date_of_incident > date_reported THEN 'INCIDENT DATED AFTER IT WAS
REPORTED' END), ''), 'complete') AS status
    FROM i ORDER BY row_no"""

```

Step 5. The register against a documented context of use

MP-1.1-002 asks for the expected and acceptable context of use to be determined and documented, with domain experts. A script cannot tell whether a context of use is good. It can tell that one is written down. Each row gets the first status that applies: no id, no context of use, no intended purpose, no owner, no risk tier, no review date, a review date not written YYYY-MM-DD, or a review older than REVIEW_MAX_DAYS before REVIEW_AS_OF. A row with none of these reads *complete*, which says only that the fields are filled and the review is recent. Owner, risk tier and review date are our checks. NIST's action asks for a documented context of use.

HR and people use cases get a separate mark. A use case is marked *FOR LEGAL TO CHECK* if its department is one of HR_DEPARTMENTS or its name or purpose holds one of HR_TERMS, in any case. The reason printed is that laws on automated employment decisions, such as NYC Local Law 144, Colorado SB 26-189 (we have not verified the bill text) and California's automated decisionmaking technology rules under the CCPA, may apply, and that other laws may also apply. Local Law 144 restricts how employers use automated employment decision tools ([NYC DCWP: Automated Employment Decision Tools](#) (read 2026-10-31)), and the California Privacy Protection Agency lists adopted regulations on automated decisionmaking technology ([California Privacy Protection Agency: regulations](#) (read 2026-10-31)). The mark is a list for your lawyers. It is not a rating, and it does not say a use case is covered by any of these laws. Any term in HR_TERMS that matches no row is printed as information, so a word that matches nothing does not pass unseen.

```

hr_terms = " OR ".join(f"contains({uc_text}, {sql_text(t)})" for t in terms) or "false"
hr_depts = ", ".join(sql_text(d) for d in HR_DEPARTMENTS if d.strip()) or "NULL"
gaps = " ".join(f"WHEN {k} IS NULL THEN '{label}'" for k, label in UC_REQUIRED)
sql = f"""
    SELECT row_no, coalesce(use_case_id_shown, '-') AS use_case_id, last_reviewed,
        CASE {gaps}
        WHEN review_text = '' THEN 'NO REVIEW DATE'
        WHEN last_reviewed IS NULL THEN 'NOT WRITTEN YYYY-MM-DD: last reviewed'
        WHEN last_reviewed < DATE '{REVIEW_AS_OF}' - {REVIEW_MAX_DAYS} THEN 'REVIEW

```

```
OVERDUE'
        ELSE 'complete' END AS status,
        CASE WHEN lower(trim(department)) IN ({hr_depts}) OR {hr_terms} THEN 'FOR LEGAL TO
CHECK' ELSE '-' END AS legal
FROM register ORDER BY row_no"""
```

Step 6. Recording by day, and a fingerprint on each list

The last list has one row for each day of the period with the number of records dated that day. A day with none before COMPLIANCE_API_ENABLED_ON is BEFORE ENABLED, because Anthropic does not backfill; any other day with none is NO RECORDS. A quiet day can look like an outage, and a day with some records but not all is invisible to this check. Anthropic says activity volume is not a completeness check ([Anthropic: Delivery guarantees and completeness](#) (read 2026-10-31)).

```
day_sql = f"""SELECT d::DATE AS day, count(a.id) AS records,
        CASE WHEN count(a.id) > 0 THEN 'recorded'
        WHEN d::DATE < {en} THEN 'BEFORE ENABLED' ELSE 'NO RECORDS' END AS
status
        FROM generate_series DATE '{p0}', DATE '{p1}', INTERVAL 1 DAY) t(d)
LEFT JOIN activities a ON a.ts::DATE = d::DATE GROUP BY 1 ORDER BY 1"""
```

Each list is printed with its period, filters, run time, row count and a SHA-256 over every row, in the order written to its CSV. The fingerprint does not prove the rows are right. It lets anyone who re-runs the script on the same files confirm they got the same rows.

The code

The complete download, `nist_ai_rmf_claude_evidence.py`. It is also at www.datameaning.com/guides/nist-ai-rmf-evidence-for-claude/nist_ai_rmf_claude_evidence.py.

```
#!/usr/bin/env python3
"""NIST AI RMF evidence for a company that uses Claude: the incident log checked against the nine
example fields NIST lists and
the use-case register checked for context of use, in DuckDB.

python3 nist_ai_rmf_claude_evidence.py # clean generated records
python3 nist_ai_rmf_claude_evidence.py --plant # generated records with planted
faults
python3 nist_ai_rmf_claude_evidence.py --short-pull # clean records, pulled from the
period's start
python3 nist_ai_rmf_claude_evidence.py --enablement-unknown # clean records, enablement day
not set
python3 nist_ai_rmf_claude_evidence.py --strict # exit 1 when anything needs
follow-up

Builds three lists, each printed with its period, filters, run time, row count and a SHA-256 of its
rows:
incident log GOVERN 4.3, action GV-4.3-002: every row of your incident log, checked for the
nine example fields NIST's
AI 600-1 lists (MISSING), dates not written YYYY-MM-DD, an incident dated
```

after it was reported, and a System ID that is not on your use-case register

use-case register MAP 1.1, action MP-1.1-002: every register row, checked for a context of use, an intended purpose, an owner, an id and a review no older than REVIEW_MAX_DAYS; HR and people use cases are also marked FOR LEGAL TO CHECK (a list for Legal, never a rating)

daily recording GOVERN 1.5 / MANAGE 4.1 (monitoring): one row per day of the period: recorded, NO RECORDS or BEFORE ENABLED, with the Compliance API's start day checked

Before it builds them it pins both NIST documents (size and SHA-256) and runs the input checks, and marks the lists INCOMPLETE if one fails; a date in your incident log or register not written YYYY-MM-DD fails one too. The record checks cover every activity record up to the period's end; faults in records after it are printed as [info] only. The check for days with no records runs from the day the Compliance API was turned on when that is before the period; days before the period listed in KNOWN_GAP_DAYS were explained in an earlier period.

Without COMPLIANCE_API_ENABLED_ON the pull cannot be checked, so that check fails.

The generated records are NOT a real organization's data. Activity Feed field names (id, created_at, type) come from Anthropic's Activity object: <https://platform.claude.com/docs/en/manage-claude/compliance-activity-feed>

NIST's field names and action texts: <https://nvlpubs.nist.gov/nistpubs/ai/NIST.AI.600-1.pdf> (July 2024) and <https://nvlpubs.nist.gov/nistpubs/ai/NIST.AI.100-1.pdf> (AI RMF 1.0). The framework is voluntary: this script lists what you can show, not whether you follow it.

```

import csv
import datetime as dt
import hashlib
import http.client
import json
import pathlib
import random
import re
import sys
import urllib.request

```

```
import duckdb
```

```

# ---- CHANGE THESE FOR YOUR OWN DATA -----
PERIOD_START = "2026-07-06"      # first day of the period you are evidencing (UTC, inclusive)
PERIOD_END = "2026-09-27"      # last day of the period (UTC, inclusive)
COMPLIANCE_API_ENABLED_ON = None # the day you first turned the Compliance API on, YYYY-MM-DD;
None fails a check
KNOWN_GAP_DAYS = []            # days BEFORE the period with no records that you explained
earlier, YYYY-MM-DD
ACTIVITIES_FILE = None         # your Activity Feed pull, one JSON object per line; None =
generate
INCIDENTS_FILE = None          # your AI incident log (CSV with a header row); None =
generate
USE_CASES_FILE = None          # your Claude use-case register (CSV with a header row); None
= generate
REVIEW_AS_OF = "2026-09-27"    # the day review ages are counted from

```

```

REVIEW_MAX_DAYS = 180                # your own policy: a register row not reviewed within this
many days is overdue
# the column of YOUR incident log that holds each of NIST's nine fields (the defaults are NIST's
own words)
INC_COLS = {
    "system_id": "System ID", "title": "Title", "reporter": "Reporter", "system_source":
"System/Source",
    "date_reported": "Data Reported", "date_of_incident": "Date of Incident", "description":
"Description",
    "impacts": "Impact(s)", "stakeholders": "Stakeholder(s) Impacted",
}
# the column of YOUR register that holds each of its fields
UC_COLS = {
    "use_case_id": "use_case_id", "use_case": "use_case", "department": "department",
    "intended_purpose": "intended_purpose", "context_of_use": "context_of_use", "owner": "owner",
    "risk_tier": "risk_tier", "last_reviewed": "last_reviewed",
}
# HR and people use cases are listed for Legal; change the words to fit your register
HR_TERMS = ["hiring", "recruit", "resume", "candidate", "promotion", "performance review",
"termination",
    "employee monitoring"]                # matched inside use_case and
intended_purpose, any case
HR_DEPARTMENTS = ["hr", "human resources", "people", "talent"] # matched against the whole
department value, any case
LEGAL_REASON = ("laws on automated employment decisions, such as NYC Local Law 144, Colorado SB 26-
189 (bill text not "
    "verified) and California's automated decisionmaking technology rules under the
CCPA, may apply. "
    "Other laws may also apply")
# -----

# NIST's own wording: the nine fields of GV-4.3-002 (the label is NIST's, including "Data
Reported")
INC_LABELS = {"system_id": "System ID", "title": "Title", "reporter": "Reporter", "system_source":
"System/Source",
    "date_reported": "Data Reported", "date_of_incident": "Date of Incident",
"description": "Description",
    "impacts": "Impact(s)", "stakeholders": "Stakeholder(s) Impacted"}
INC_DATES = ("date_reported", "date_of_incident")
UC_REQUIRED = [("use_case_id", "NO USE CASE ID"), ("context_of_use", "NO CONTEXT OF USE"),
    ("intended_purpose", "NO INTENDED PURPOSE"), ("owner", "NO OWNER"), ("risk_tier",
"NO RISK TIER")]
# the two NIST documents, pinned: a different size or hash means NIST re-published the file
NIST_DOCS = [
    ("NIST AI 600-1 (Generative AI Profile, July 2024)",
    "https://nvlpubs.nist.gov/nistpubs/ai/NIST.AI.600-1.pdf",
    1174643, "6e73620ab6b64e90ef2c04bf0e0d6246185a2f4b1b13cab0df494496cff89b6a"),
    ("NIST AI 100-1 (AI RMF 1.0, January 2023)", "https://nvlpubs.nist.gov/nistpubs/ai/NIST.AI.100-
1.pdf",
    1946127, "7576edb531d9848825814ee88e28b1795d3a84b435b4b797d3670eafdc4a89f1"),
]
USER_AGENT = ("nist_ai_rmf_claude_evidence.py (https://www.datameaning.com/guides/nist-ai-rmf-
evidence-for-claude/; "
    "info@datameaning.com)")
GEN_ENABLED = "2026-06-01"            # generated records only: the day the stand-in organization
turned the API on
OUT_DIR = pathlib.Path("evidence-out")
GEN_DIR = pathlib.Path("generated")

```

```

DATE_FORM = r"\d{4}-\d{2}-\d{2}"

def require_date(name, value):
    """Dates are pasted into SQL, so each one must be a real YYYY-MM-DD before it is used."""
    try:
        if not re.fullmatch(DATE_FORM, value):
            raise ValueError
        dt.date.fromisoformat(value)
    except (TypeError, ValueError):
        sys.exit(f"{name} must be a date written YYYY-MM-DD, not {value!r}")

def qcol(name):
    return "'" + name.replace("'", '""') + "'"

def generate(mode):
    """Seeded stand-in for an organization's feed, incident log and register. mode: clean, plant or short."""
    plant = mode == "plant"
    rnd = random.Random(224)
    GEN_DIR.mkdir(exist_ok=True)
    users = [f"u{n:02d}@example.test" for n in range(1, 41)]
    start, end = dt.date.fromisoformat(GEN_ENABLED), dt.date.fromisoformat(PERIOD_END) +
dt.timedelta(days=3)
    recs, day = [], start
    while day <= end:
        for typ in ["sso_login_succeeded"] + [rnd.choice(["claude_chat_created",
"claude_chat_viewed"])]
                                for _ in range(rnd.randrange(3, 8))]:
            t = dt.datetime.combine(day, dt.time(rnd.randrange(7, 19), rnd.randrange(60),
rnd.randrange(60)))
            who = rnd.choice(users)
            recs.append({"id": "", "created_at": t.strftime("%Y-%m-%dT%H:%M:%SZ"),
"organization_id": "org_generated",
                        "actor": {"type": "user_actor", "email_address": who, "user_id": "user_" +
who.split("@")[0]},
                        "type": typ})
            day += dt.timedelta(days=1)
        enabled = GEN_ENABLED
        if plant:
            enabled = "2026-07-08" # planted: Compliance
API turned on two days late
            recs = [r for r in recs if r["created_at"][:10] >= enabled
                    and not "2026-08-10" <= r["created_at"][:10] <= "2026-08-14"] # planted: five days
with no records
        if mode == "short":
            recs = [r for r in recs if r["created_at"][:10] >= PERIOD_START] # planted: pulled from
the period, not enablement
            recs.sort(key=lambda r: (r["created_at"], r["type"], json.dumps(r["actor"], sort_keys=True)))
            for n, r in enumerate(recs, 1):
                r["id"] = f"activity_{n:06d}"
            if plant:
                noise = [i for i, r in enumerate(recs) if r["type"] == "claude_chat_viewed" and i > 100]
                recs[noise[0]]["created_at"] = "not-a-date" # planted: an
unreadable timestamp
                recs.append(dict(recs[noise[1]])) # planted: a

```

```

duplicated record
    after = [i for i, r in enumerate(recs) if r["created_at"][:10] > PERIOD_END]
    recs.append(dict(recs[after[1]])) # a duplicate after
the period's end:
    # not the period's fault, so it is reported as [info] and the period's checks stay clean
    feed = GEN_DIR / f"feed_{mode}.jsonl"
    feed.write_text(''.join(json.dumps(r) + "\n" for r in recs))

    lead, ops = "support.lead@example.test", "data.platform@example.test"
    # the risk tiers below are invented for the demo; they are not a rating of any real use case
    reg = [{"UC-001", "Summarize closed support tickets", "Customer Support", "Draft summaries of
closed tickets for team leads",
            "Internal staff only; a lead reads every summary before it is reused", lead, "medium",
"2026-06-15"],
          ["UC-002", "Draft and edit marketing copy", "Marketing", "First drafts of web and email
copy",
            "Internal drafts; a person approves anything published", "marketing.lead@example.test",
"low", "2026-07-02"],
          ["UC-003", "Explain and review SQL", "Data Platform", "Explain queries and suggest fixes
to engineers",
            "Engineers only; no production data pasted; changes go through code review", ops,
"low", "2026-08-20"],
          ["UC-004", "Look up contract clauses", "Legal", "Find and compare clauses in contracts
Legal already holds",
            "Legal staff only; output is a starting point for a lawyer", "legal.ops@example.test",
"medium", "2026-05-28"],
          ["UC-005", "Screen resumes for open roles", "HR", "Rank resumes against a job
description for recruiters",
            "Recruiters only; a recruiter makes every decision", "hr.lead@example.test", "high",
"2026-09-01"],
          ["UC-006", "Answer finance policy questions", "Finance", "Answer staff questions from
the finance policy library",
            "Internal staff only; answers cite the policy page", "finance.ops@example.test", "low",
"2026-07-30"]]
    inc = [{"UC-001", "Summary quoted a line from the wrong ticket", "support.agent@example.test",
"Claude Enterprise, Support workspace",
            "2026-07-20", "2026-07-18", "A summary included one sentence from a different ticket.",
            "Wrong text reached a team lead; caught before reuse", "Customer Support leads"],
          ["UC-003", "Suggested query dropped a filter", "data.eng@example.test", "Claude
Enterprise, Data Platform project",
            "2026-08-12", "2026-08-11", "A suggested query ran without its date filter in a test.",
            "Test results were wrong; no production data changed", "Data engineers"],
          ["UC-002", "Draft stated an unreleased feature as shipped",
"marketing.writer@example.test", "Claude Enterprise, Marketing project",
            "2026-09-03", "2026-09-02", "A draft described a feature that was not released.",
            "Draft corrected before publication", "Marketing team"]]
    if plant:
        by = {r[0]: r for r in reg}
        by["UC-003"][4] = "" # planted: no context
of use
        by["UC-002"][5] = "" # planted: no owner
        by["UC-006"][3] = "" # planted: no intended
purpose
        by["UC-004"][7] = "2025-11-02" # planted: review
overdue
        by["UC-001"][7] = "6/15/26" # planted: a short
date, which loose parsers read as year 6
        by["UC-005"][7] = "" # planted: no review

```

```

date at all
    reg += [{" uc-007 ", "Rank internal candidates for promotion", "People", "Suggest which
staff to put forward for promotion",
            "HR only; calibrated by managers", "people.ops@example.test", "high", "2026-08-
15"}, # planted: another HR row; id in other case with spaces
            ["UC-003", "Explain and review SQL", "Data Platform", "Explain queries", "Engineers
only", ops, "low", "2026-08-20"], # planted: duplicate id
            ["", "Untracked chat helper", "Operations", "Unknown", "Unknown", ops, "low",
"2026-08-20"]] # planted: no id
    inc = [list(r) for r in inc]
    inc[0][2] = "" # planted: no reporter
    ...
    inc[0][7] = "" # ... and no impacts
    inc[1][5] = "2026/08/11" # planted: date of
incident not YYYY-MM-DD
    inc += [{"UC-099", "Chat helper gave wrong refund rule", "support.agent@example.test",
"Claude Enterprise",
            "2026-08-25", "2026-08-24", "Wrong refund rule quoted.", "Customer told the wrong
rule", "Customers"}, # planted: use case not on register
            ["UC-006", "Policy answer cited a retired page", "finance.user@example.test",
"Claude Enterprise, Finance project",
            "2026-09-01", "2026-09-10", "Answer cited a retired policy page.", "Staff read old
guidance", "Finance staff"], # planted: incident after report
            ["", "Output with no known use case", "it.ops@example.test", "Claude Enterprise",
"2026-09-15", "2026-09-14", "Unclear output.", "Unknown", "Unknown"], # planted:
no System ID
            ["UC-004", "Clause comparison missed a carve-out", "legal.ops@example.test",
"Claude Enterprise, Legal project",
            "8/5/26", "2026-08-04", "A comparison missed one clause.", "Lawyer caught it",
"Legal staff"], # planted: Data Reported not YYYY-MM-DD
            list(inc[2]), # planted: a
duplicated incident row
            [{" uc-001 ", "Summary mixed two tickets", "support.agent@example.test", "Claude
Enterprise, Support workspace",
            "2026-09-18", "2026-09-17", "Two tickets merged in a summary.", "Lead noticed
before reuse", "Customer Support leads"]} # same id, other case: still matches
    with open(GEN_DIR / "use_cases.csv", "w", newline="") as f:
        w = csv.writer(f)
        w.writerow(["use_case_id", "use_case", "department", "intended_purpose", "context_of_use",
"owner", "risk_tier", "last_reviewed"])
        w.writerows(reg)
    with open(GEN_DIR / "incidents.csv", "w", newline="") as f:
        w = csv.writer(f)
        w.writerow([INC_LABELS[k] for k in INC_LABELS])
        w.writerows(inc)
    return str(feed), str(GEN_DIR / "incidents.csv"), str(GEN_DIR / "use_cases.csv"), enabled

def fingerprint(rows):
    body = "\n".join("|".join(" " if v is None else str(v) for v in r) for r in rows)
    return hashlib.sha256(body.encode()).hexdigest()

def population(con, name, control, filters, sql, header, complete):
    rows = con.execute(sql).fetchall()
    OUT_DIR.mkdir(exist_ok=True)
    with open(OUT_DIR / f"{name}.csv", "w", newline="") as f:
        w = csv.writer(f)

```

```

        w.writerow(header)
        w.writerows(rows)
    print(f"\n== {control}: {name}" + (" " if complete else "  ** INCOMPLETE: see input checks
**"))
    print(f"period      : {PERIOD_START} to {PERIOD_END} (UTC, inclusive)")
    print(f"filters       : {filters}")
    print(f"run at        : {dt.datetime.now(dt.timezone.utc):%Y-%m-%d %H:%M:%S} UTC")
    print(f"rows         : {len(rows)}")
    print(f"sha256        : {fingerprint(rows)}")
    return rows

def ranges(days):
    """Collapse sorted dates into (first, last, count) stretches."""
    out = []
    for d in days:
        if out and (d - out[-1][1]).days == 1:
            out[-1][1] = d
            out[-1][2] += 1
        else:
            out.append([d, d, 1])
    return out

def strict_date(col):
    """YYYY-MM-DD only. DuckDB's own cast is lenient: it reads 8/5/26 as the year 8."""
    c = f"trim(coalesce({qcol(col)}, ''))"
    return f"CASE WHEN regexp_full_match({c}, '{DATE_FORM}') THEN try_strptime({c}, '%Y-%m-%d')::DATE END"

def sql_text(value):
    """A word from the settings as a SQL string literal: trimmed, lower-cased, quotes doubled."""
    return "'" + value.strip().lower().replace("'", "'") + "'"

def show(rows):
    for r in rows:
        print(" " + " | ".join("-" if v is None else str(v) for v in r))

def pin_nist():
    """Download each NIST document once, print its size, Last-Modified and SHA-256, return the
    number that do not match."""
    bad = 0
    for label, url, size, sha in NIST_DOCS:
        try:
            req = urllib.request.Request(url, headers={"User-Agent": USER_AGENT})
            with urllib.request.urlopen(req, timeout=60) as r:
                body, modified = r.read(), r.headers.get("Last-Modified", "not sent")
            except (OSError, http.client.HTTPException) as e:
                # a cut-off download is a failed
                pin, not a crash
                print(f" {label}: could not be fetched ({e.__class__.__name__}); the pin cannot be
checked")
                bad += 1
                continue
            got = hashlib.sha256(body).hexdigest()
            same = len(body) == size and got == sha

```

```

        bad += 0 if same else 1
        print(f" {label}: {len(body)} bytes, Last-Modified {modified}, sha256 {got}" + ("'" if same
else "    ** DIFFERENT FROM THE PIN **"))
    return bad

def main():
    strict = "--strict" in sys.argv
    mode = "plant" if "--plant" in sys.argv else "short" if "--short-pull" in sys.argv else "clean"
    acts, inc_file, uc_file, enabled = ACTIVITIES_FILE, INCIDENTS_FILE, USE_CASES_FILE,
COMPLIANCE_API_ENABLED_ON
    for name, v in [("PERIOD_START", PERIOD_START), ("PERIOD_END", PERIOD_END), ("REVIEW_AS_OF",
REVIEW_AS_OF)]:
        require_date(name, v)
    for d in KNOWN_GAP_DAYS:
        require_date("KNOWN_GAP_DAYS", d)
    if not isinstance(REVIEW_MAX_DAYS, int) or REVIEW_MAX_DAYS <= 0:
        sys.exit("REVIEW_MAX_DAYS must be a whole number of days above zero")
    if acts is None:
        acts, inc_file, uc_file, enabled = generate(mode)
        if "--enablement-unknown" in sys.argv:
            enabled = None                                # planted: the
enablement day not set
        print(f"input      : GENERATED records (not a real organization), seed 224, mode={mode}")
    else:
        if inc_file is None or uc_file is None:
            sys.exit("set INCIDENTS_FILE and USE_CASES_FILE as well as ACTIVITIES_FILE")
        print("input      : " + acts + ", " + inc_file + ", " + uc_file)
    if enabled is not None:
        require_date("COMPLIANCE_API_ENABLED_ON", enabled)
    print("enabled on: " + (enabled or "not set (COMPLIANCE_API_ENABLED_ON is None)"))
    print("\n== NIST documents (pinned: a different size or hash means NIST re-published the
file)")
    nist_bad = pin_nist()

    con = duckdb.connect()
    con.execute("CREATE TABLE raw AS SELECT * FROM read_json(?, format='newline_delimited', "
                "columns={ 'id': 'VARCHAR', 'created_at': 'VARCHAR', 'type': 'VARCHAR' })", [acts])
    con.execute("""CREATE TABLE activities AS SELECT id, type, try_cast(replace(created_at, 'Z',
'') AS TIMESTAMP) AS ts FROM raw""")
    # every column your settings name must exist in your files; a wrong name stops the run with the
names it did find
    for label, path, cols in [("incident log", inc_file, INC_COLS), ("use-case register", uc_file,
UC_COLS)]:
        have = {r[0].lower() for r in con.execute("DESCRIBE SELECT * FROM read_csv(?, header=true,
all_varchar=true)", [path]).fetchall()}
        lost = [c for c in cols.values() if c.lower() not in have]
        if lost:
            sys.exit(f"{label}: no column named {'', '.join(lost)} in {path}; its columns are {'',
'.join(sorted(have))}")
        txt = lambda c: f"nullif(trim(coalesce({qcol(c)}, '')), '')"
        inc_sel = ", ".join(f"{txt(INC_COLS[k])} AS {k}" for k in INC_LABELS if k not in INC_DATES)
        inc_sel += ", " + ", ".join(f"{strict_date(INC_COLS[k])} AS {k},
trim(coalesce({qcol(INC_COLS[k]}), '')) AS {k}_text" for k in INC_DATES)
        con.execute(f""""CREATE TABLE incidents AS SELECT row_number() OVER () AS row_no, {inc_sel}
FROM read_csv(?, header=true, all_varchar=true)""", [inc_file])
        uc_sel = (f"nullif(lower(trim(coalesce({qcol(UC_COLS['use_case_id'])), ''))), '') AS
use_case_id, "

```

```

        f"{txt(UC_COLS['use_case_id'])} AS use_case_id_shown, "
        + ", ".join(f"{txt(UC_COLS[k])} AS {k}" for k in ("use_case", "department",
"intended_purpose", "context_of_use", "owner", "risk_tier"))
        + f", {strict_date(UC_COLS['last_reviewed'])} AS last_reviewed,
trim(coalesce({qcol(UC_COLS['last_reviewed'])}, '')) AS review_text")
    con.execute(f"CREATE TABLE register AS SELECT row_number() OVER () AS row_no, {uc_sel} FROM
read_csv(?, header=true, all_varchar=true)", [uc_file])
    p0, p1 = PERIOD_START, PERIOD_END
    en = f"DATE '{enabled}'" if enabled else "NULL::DATE"
    day_sql = f""""SELECT d::DATE AS day, count(a.id) AS records,
CASE WHEN count(a.id) > 0 THEN 'recorded'
WHEN d::DATE < {en} THEN 'BEFORE ENABLED' ELSE 'NO RECORDS' END AS
status
FROM generate_series(DATE '{p0}', DATE '{p1}', INTERVAL 1 DAY) t(d)
LEFT JOIN activities a ON a.ts::DATE = d::DATE GROUP BY 1 ORDER BY 1""""

# ---- input checks: each says which planted fault trips it
q = lambda s: con.execute(s).fetchone()[0]
upto = f"(ts IS NULL OR ts::DATE <= DATE '{p1}')"
later = f"ts::DATE > DATE '{p1}'"
total = q("SELECT count(*) FROM activities")
before = q(f"SELECT count(*) FROM activities WHERE ts::DATE < DATE '{p0}'")
after = q(f"SELECT count(*) FROM activities WHERE {later}")
earliest = q("SELECT min(ts)::DATE FROM activities")
days = con.execute(day_sql).fetchall()
pre = [r[0] for r in days if r[2] == "BEFORE ENABLED"]
# days with no records, from the day the Compliance API was turned on (or the period's start
when that is not known)
scan_from = enabled if enabled and enabled < p0 else p0
off = [r[0] for r in con.execute(f""""SELECT d::DATE FROM generate_series(DATE '{scan_from}',
DATE '{p1}', INTERVAL 1 DAY) t(d)
LEFT JOIN activities a ON a.ts::DATE = d::DATE
WHERE {en} IS NULL OR d::DATE >= {en} GROUP BY 1 HAVING
count(a.id) = 0 ORDER BY 1""").fetchall()]]
known = set(KNOWN_GAP_DAYS)
explained = [d for d in off if str(d) < p0 and str(d) in known]
off = [d for d in off if d not in explained]
# not evaluated is a failure: without the enablement day the pull cannot be checked. Quiet days
right after
# enablement that were explained earlier (KNOWN_GAP_DAYS) move the day the pull must reach back
to forward.
reach_to = enabled
if enabled:
    d = dt.date.fromisoformat(enabled)
    while str(d) in known and str(d) < p0:
        d += dt.timedelta(days=1)
    reach_to = str(d)
short_pull = 1 if not enabled or (earliest and str(earliest) > reach_to) else 0
reach_label = ("the pull does not reach back to the day the Compliance API was turned on" if
enabled else
               "the day the Compliance API was turned on is not set, so the pull cannot be
checked")
    bad_dates = (q("SELECT count(*) FROM incidents WHERE " + " OR ".join(f"({k}_text <> '' AND {k}
IS NULL)" for k in INC_DATES))
                + q("SELECT count(*) FROM register WHERE review_text <> '' AND last_reviewed IS
NULL"))
    checks = [
        ("NIST documents that are not the pinned file", nist_bad, "a NIST file re-published, or a

```

```

fetch that failed"),
    ("duplicate activity ids", q(f"SELECT count(*) FROM (SELECT id FROM activities WHERE {upto}
GROUP BY id HAVING count(*) > 1)"), "a duplicated record"),
    ("activity records with no id or no readable created_at", q(f"SELECT count(*) FROM
activities WHERE (ts IS NULL OR id IS NULL) AND {upto}"), "an unreadable timestamp"),
    (reach_label, short_pull, "a pull that starts at the period; the enablement day not set"),
    ("days with no records, from enablement (or the period's start) to its end", len(off),
"days missing from the pull; a pull that starts at the period"),
    ("days in the period before the Compliance API was enabled", len(pre), "the Compliance API
turned on late"),
    ("incident log rows repeated (same System ID, Title and Date of Incident)",
q("SELECT count(*) FROM (SELECT 1 FROM incidents WHERE title IS NOT NULL GROUP BY
lower(system_id), lower(title), date_of_incident_text HAVING count(*) > 1)"),
    "a duplicated incident row"),
    ("register ids listed more than once", q("SELECT count(*) FROM (SELECT use_case_id FROM
register WHERE use_case_id IS NOT NULL GROUP BY 1 HAVING count(*) > 1)"), "a duplicated register
row"),
    ("register rows with no use case id", q("SELECT count(*) FROM register WHERE use_case_id IS
NULL"), "a register row with no id"),
    ("incident log and register dates not written YYYY-MM-DD", bad_dates, "a short date such as
8/5/26"),
    ]
    n_inc, n_uc = q("SELECT count(*) FROM incidents"), q("SELECT count(*) FROM register")
    print(f"\n== Input checks    (activity records read: {total}; before the period: {before}; after
the period's end: {after}, not used;")
        f" incident log rows: {n_inc}; register rows: {n_uc}))")
    ok = True
    for label, n, fault in checks:
        ok &= n == 0
        print(f"[{'ok' if n == 0 else 'FAIL'}] {label}: {n}" + (" (trips on:
{fault})"))
    print(f"    earliest record: {earliest}; enabled on: {enabled or 'not set'}"
        + (f"; must reach back to: {reach_to} (explained days at the start skipped)" if enabled
and reach_to != enabled else ""))
    if explained:
        print("[info] days before the period with no records, explained earlier (KNOWN_GAP_DAYS): "
+ ", ".join(str(d) for d in explained))
        inside = sorted(d for d in known if p0 <= d <= p1)
        if inside:
            print("[info] KNOWN_GAP_DAYS inside the period are not accepted, they still count: " + ",
".join(inside))
        for label, ds in [("days with no records", off), ("days before enablement", pre)]:
            if ds:
                print(f"    {label}: " + ", ".join(f"{a}..{b}" if a != b else str(a) for a, b, _ in
ranges(ds)))
    print(f"[info] after the period's end, not checked: duplicate activity ids "
        f"{q(f'SELECT count(*) FROM (SELECT id FROM activities WHERE {later} GROUP BY id HAVING
count(*) > 1)')}"); "
        f"records with no id {q(f'SELECT count(*) FROM activities WHERE id IS NULL AND
{later}')}")
    in_period = q(f"SELECT count(*) FROM incidents WHERE date_of_incident BETWEEN DATE '{p0}' AND
DATE '{p1}'")
    no_date = q("SELECT count(*) FROM incidents WHERE date_of_incident IS NULL")
    print(f"[info] incidents with a Date of Incident inside the period: {in_period} of {n_inc}"
        f" ({no_date} with no Date of Incident written YYYY-MM-DD, so not placed)")

# ---- GOVERN 4.3 / GV-4.3-002: the incident log against NIST's nine fields
missing = ", ".join(f"CASE WHEN {k}{'_text' if k in INC_DATES else ''} {'=' + chr(39) * 2 if k

```

```

in INC_DATES else 'IS NULL'} THEN '{INC_LABELS[k]}' END"
    for k in INC_LABELS)
unread = ", ".join(f"CASE WHEN {k}_text <> '' AND {k} IS NULL THEN '{INC_LABELS[k]}' END" for k
in INC_DATES)
sql = f"""
    WITH reg AS (SELECT DISTINCT use_case_id FROM register WHERE use_case_id IS NOT NULL),
    i AS (SELECT n.*, concat_ws(' ', {missing}) AS missing, concat_ws(' ', {unread}) AS
unread,
        r.use_case_id IS NULL AS off_register
    FROM (SELECT *, lower(system_id) AS sid FROM incidents) n LEFT JOIN reg r ON
r.use_case_id = n.sid)
    SELECT row_no, coalesce(system_id, '-') AS system_id, date_of_incident,
        coalesce(nullif(concat_ws(' ',
            CASE WHEN missing <> '' THEN 'MISSING: ' || missing END,
            CASE WHEN unread <> '' THEN 'NOT WRITTEN YYYY-MM-DD: ' || unread END,
            CASE WHEN system_id IS NOT NULL AND off_register THEN 'SYSTEM NOT ON REGISTER'
END,
            CASE WHEN date_of_incident > date_reported THEN 'INCIDENT DATED AFTER IT WAS
REPORTED' END), ''), 'complete') AS status
    FROM i ORDER BY row_no"""
r_inc = population(con, "incident_log_check", "GOVERN 4.3 / GV-4.3-002",
    "every row of the incident log, none dropped; row_no is the row's place in
the file after the header; "
    "System ID matched to the register on the lower-cased, trimmed id; the nine
fields as NIST's AI 600-1 lists them",
    sql, ["row_no", "system_id", "date_of_incident", "status"], ok)
show(r_inc)

# ---- MAP 1.1 / MP-1.1-002: the register against a documented context of use
terms = [t for t in HR_TERMS if t.strip()]
uc_text = "lower(coalesce(use_case, '')) || ' ' || coalesce(intended_purpose, '')"
hr_terms = " OR ".join(f"contains({uc_text}, {sql_text(t)})" for t in terms) or "false"
hr_depts = ", ".join(sql_text(d) for d in HR_DEPARTMENTS if d.strip()) or "NULL"
gaps = " ".join(f"WHEN {k} IS NULL THEN '{label}'" for k, label in UC_REQUIRED)
sql = f"""
    SELECT row_no, coalesce(use_case_id_shown, '-') AS use_case_id, last_reviewed,
        CASE {gaps}
            WHEN review_text = '' THEN 'NO REVIEW DATE'
            WHEN last_reviewed IS NULL THEN 'NOT WRITTEN YYYY-MM-DD: last reviewed'
            WHEN last_reviewed < DATE '{REVIEW_AS_OF}' - {REVIEW_MAX_DAYS} THEN 'REVIEW
OVERDUE'
            ELSE 'complete' END AS status,
        CASE WHEN lower(trim(department)) IN ({hr_depts}) OR {hr_terms} THEN 'FOR LEGAL TO
CHECK' ELSE '-' END AS legal
    FROM register ORDER BY row_no"""
r_uc = population(con, "use_case_register_check", "MAP 1.1 / MP-1.1-002",
    f"every register row, none dropped; first applicable status; review overdue
means last reviewed before {REVIEW_AS_OF} minus {REVIEW_MAX_DAYS} days; "
    "FOR LEGAL TO CHECK marks HR and people use cases by your HR_TERMS and
HR_DEPARTMENTS",
    sql, ["row_no", "use_case_id", "last_reviewed", "status", "legal"], ok)
show(r_uc)
legal_n = sum(1 for r in r_uc if r[4] != "-")
print(f" for Legal to check: {legal_n} use case(s). Reason: {LEGAL_REASON}. This is a list for
Legal, not a rating.")
dead = [t for t in terms if q(f"SELECT count(*) FROM register WHERE contains({uc_text},
{sql_text(t)})") == 0]
if dead:

```

```

        print("[info] HR_TERMS that matched no register row: " + ", ".join(dead))

# ---- GOVERN 1.5 / MANAGE 4.1: was the feed recording on every day of the period?
r_log = population(con, "recording_daily", "GOVERN 1.5 / MANAGE 4.1",
                  "one row per day; records counted by created_at date; enabled on " +
(enabled or "not set"), day_sql,
                  ["day", "records", "status"], ok)
print(f"  days recorded: {sum(1 for r in r_log if r[2] == 'recorded')} of {len(r_log)}")
for st in ("NO RECORDS", "BEFORE ENABLED"):
    for a, b, n in ranges([r[0] for r in r_log if r[2] == st]):
        print(f"  {st}: {a} to {b} ({n} day(s))")
print(f"  from {scan_from} to the period's end, days with no records: {len(off)}")

follow = (sum(1 for r in r_inc if r[3] != "complete") + sum(1 for r in r_uc if r[3] !=
"complete") + len(off) + len(pre))
print("\n== Result")
print(f"input checks : {'all passed' if ok else 'FAILED, the lists above are not complete'}")
print(f"follow-up      : {follow} item(s) need an explanation "
      f"(incident rows and register rows with a gap, days with no records or before
enablement)")
verdict = "NO GAPS FOUND" if ok and follow == 0 else "GAPS FOUND"
print(f"RESULT: {verdict}")
return 1 if strict and verdict != "NO GAPS FOUND" else 0

if __name__ == "__main__":
    sys.exit(main())

```

The runs

Each run is the script's own output, run on generated records on 2026-10-31.

Run 1: planted faults

Command: `python3 nist_ai_rmf_claude_evidence.py --plant`

```

input      : GENERATED records (not a real organization), seed 224, mode=plant
enabled on: 2026-07-08

== NIST documents (pinned: a different size or hash means NIST re-published the file)
  NIST AI 600-1 (Generative AI Profile, July 2024): 1174643 bytes, Last-Modified Mon, 24 Mar 2025
19:11:27 GMT, sha256 6e73620ab6b64e90ef2c04bf0e0d6246185a2f4b1b13cab0df494496cfff89b6a
  NIST AI 100-1 (AI RMF 1.0, January 2023): 1946127 bytes, Last-Modified Wed, 04 Jun 2025 17:14:26
GMT, sha256 7576edb531d9848825814ee88e28b1795d3a84b435b4b797d3670eafdc4a89f1

== Input checks  (activity records read: 457; before the period: 0; after the period's end: 16,
not used; incident log rows: 9; register rows: 9)
[ok ] NIST documents that are not the pinned file: 0
[FAIL] duplicate activity ids: 1  (trips on: a duplicated record)
[FAIL] activity records with no id or no readable created_at: 1  (trips on: an unreadable
timestamp)
[ok ] the pull does not reach back to the day the Compliance API was turned on: 0
[FAIL] days with no records, from enablement (or the period's start) to its end: 5  (trips on:
days missing from the pull; a pull that starts at the period)

```

```

[FAIL] days in the period before the Compliance API was enabled: 2 (trips on: the Compliance API
turned on late)
[FAIL] incident log rows repeated (same System ID, Title and Date of Incident): 1 (trips on: a
duplicated incident row)
[FAIL] register ids listed more than once: 1 (trips on: a duplicated register row)
[FAIL] register rows with no use case id: 1 (trips on: a register row with no id)
[FAIL] incident log and register dates not written YYYY-MM-DD: 3 (trips on: a short date such as
8/5/26)
    earliest record: 2026-07-08; enabled on: 2026-07-08
    days with no records: 2026-08-10..2026-08-14
    days before enablement: 2026-07-06..2026-07-07
[info] after the period's end, not checked: duplicate activity ids 1; records with no id 0
[info] incidents with a Date of Incident inside the period: 8 of 9 (1 with no Date of Incident
written YYYY-MM-DD, so not placed)

== GOVERN 4.3 / GV-4.3-002: incident_log_check ** INCOMPLETE: see input checks **
period : 2026-07-06 to 2026-09-27 (UTC, inclusive)
filters : every row of the incident log, none dropped; row_no is the row's place in the file
after the header; System ID matched to the register on the lower-cased, trimmed id; the nine fields
as NIST's AI 600-1 lists them
run at : 2026-10-31 16:45:21 UTC
rows : 9
sha256 : e2c19afc3f4657655745285cb6c39f4f7681f0fb8f7af91c1b5442154a282a3e
  1 | UC-001 | 2026-07-18 | MISSING: Reporter, Impact(s)
  2 | UC-003 | - | NOT WRITTEN YYYY-MM-DD: Date of Incident
  3 | UC-002 | 2026-09-02 | complete
  4 | UC-099 | 2026-08-24 | SYSTEM NOT ON REGISTER
  5 | UC-006 | 2026-09-10 | INCIDENT DATED AFTER IT WAS REPORTED
  6 | - | 2026-09-14 | MISSING: System ID
  7 | UC-004 | 2026-08-04 | NOT WRITTEN YYYY-MM-DD: Data Reported
  8 | UC-002 | 2026-09-02 | complete
  9 | uc-001 | 2026-09-17 | complete

== MAP 1.1 / MP-1.1-002: use_case_register_check ** INCOMPLETE: see input checks **
period : 2026-07-06 to 2026-09-27 (UTC, inclusive)
filters : every register row, none dropped; first applicable status; review overdue means last
reviewed before 2026-09-27 minus 180 days; FOR LEGAL TO CHECK marks HR and people use cases by your
HR_TERMS and HR_DEPARTMENTS
run at : 2026-10-31 16:45:21 UTC
rows : 9
sha256 : ab259e44b3f26fcf62bcd1b81963c429c641b9e158e7d4b8ddc806c0310e3757
  1 | UC-001 | - | NOT WRITTEN YYYY-MM-DD: last reviewed | -
  2 | UC-002 | 2026-07-02 | NO OWNER | -
  3 | UC-003 | 2026-08-20 | NO CONTEXT OF USE | -
  4 | UC-004 | 2025-11-02 | REVIEW OVERDUE | -
  5 | UC-005 | - | NO REVIEW DATE | FOR LEGAL TO CHECK
  6 | UC-006 | 2026-07-30 | NO INTENDED PURPOSE | -
  7 | uc-007 | 2026-08-15 | complete | FOR LEGAL TO CHECK
  8 | UC-003 | 2026-08-20 | complete | -
  9 | - | 2026-08-20 | NO USE CASE ID | -
    for Legal to check: 2 use case(s). Reason: laws on automated employment decisions, such as NYC
Local Law 144, Colorado SB 26-189 (bill text not verified) and California's automated
decisionmaking technology rules under the CCPA, may apply. Other laws may also apply. This is a
list for Legal, not a rating.
[info] HR_TERMS that matched no register row: hiring, performance review, termination, employee
monitoring

== GOVERN 1.5 / MANAGE 4.1: recording_daily ** INCOMPLETE: see input checks **

```

```

period      : 2026-07-06 to 2026-09-27 (UTC, inclusive)
filters     : one row per day; records counted by created_at date; enabled on 2026-07-08
run at      : 2026-10-31 16:45:21 UTC
rows        : 84
sha256      : 03965a69e5fff5548657218403867cd91412182a61283e48cd84bb4c33d90b8b
days recorded: 77 of 84
NO RECORDS: 2026-08-10 to 2026-08-14 (5 day(s))
BEFORE ENABLED: 2026-07-06 to 2026-07-07 (2 day(s))
from 2026-07-06 to the period's end, days with no records: 5

== Result
input checks : FAILED, the lists above are not complete
follow-up    : 20 item(s) need an explanation (incident rows and register rows with a gap, days
with no records or before enablement)
RESULT: GAPS FOUND

```

Run 2: clean records

Command: `python3 nist_ai_rmf_claude_evidence.py`

```

input      : GENERATED records (not a real organization), seed 224, mode=clean
enabled on: 2026-06-01

== NIST documents (pinned: a different size or hash means NIST re-published the file)
  NIST AI 600-1 (Generative AI Profile, July 2024): 1174643 bytes, Last-Modified Mon, 24 Mar 2025
19:11:27 GMT, sha256 6e73620ab6b64e90ef2c04bf0e0d6246185a2f4b1b13cab0df494496cff89b6a
  NIST AI 100-1 (AI RMF 1.0, January 2023): 1946127 bytes, Last-Modified Wed, 04 Jun 2025 17:14:26
GMT, sha256 7576edb531d9848825814ee88e28b1795d3a84b435b4b797d3670eafdc4a89f1

== Input checks (activity records read: 694; before the period: 199; after the period's end: 16,
not used; incident log rows: 3; register rows: 6)
[ok ] NIST documents that are not the pinned file: 0
[ok ] duplicate activity ids: 0
[ok ] activity records with no id or no readable created_at: 0
[ok ] the pull does not reach back to the day the Compliance API was turned on: 0
[ok ] days with no records, from enablement (or the period's start) to its end: 0
[ok ] days in the period before the Compliance API was enabled: 0
[ok ] incident log rows repeated (same System ID, Title and Date of Incident): 0
[ok ] register ids listed more than once: 0
[ok ] register rows with no use case id: 0
[ok ] incident log and register dates not written YYYY-MM-DD: 0
      earliest record: 2026-06-01; enabled on: 2026-06-01
[info] after the period's end, not checked: duplicate activity ids 1; records with no id 0
[info] incidents with a Date of Incident inside the period: 3 of 3 (0 with no Date of Incident
written YYYY-MM-DD, so not placed)

== GOVERN 4.3 / GV-4.3-002: incident_log_check
period      : 2026-07-06 to 2026-09-27 (UTC, inclusive)
filters     : every row of the incident log, none dropped; row_no is the row's place in the file
after the header; System ID matched to the register on the lower-cased, trimmed id; the nine fields
as NIST's AI 600-1 lists them
run at      : 2026-10-31 16:45:27 UTC
rows        : 3
sha256      : d0e76435a95387a55368e26ebff452cc3eb92f861689db562fd296c18b376a81
  1 | UC-001 | 2026-07-18 | complete
  2 | UC-003 | 2026-08-11 | complete

```

```

3 | UC-002 | 2026-09-02 | complete

== MAP 1.1 / MP-1.1-002: use_case_register_check
period      : 2026-07-06 to 2026-09-27 (UTC, inclusive)
filters     : every register row, none dropped; first applicable status; review overdue means last
              reviewed before 2026-09-27 minus 180 days; FOR LEGAL TO CHECK marks HR and people use cases by your
              HR_TERMS and HR_DEPARTMENTS
run at      : 2026-10-31 16:45:27 UTC
rows        : 6
sha256      : b9e339a468c693d39c642bde8faad2bd673d71014453bc4f1cb315653ed35b69
  1 | UC-001 | 2026-06-15 | complete | -
  2 | UC-002 | 2026-07-02 | complete | -
  3 | UC-003 | 2026-08-20 | complete | -
  4 | UC-004 | 2026-05-28 | complete | -
  5 | UC-005 | 2026-09-01 | complete | FOR LEGAL TO CHECK
  6 | UC-006 | 2026-07-30 | complete | -
    for Legal to check: 1 use case(s). Reason: laws on automated employment decisions, such as NYC
    Local Law 144, Colorado SB 26-189 (bill text not verified) and California's automated
    decisionmaking technology rules under the CCPA, may apply. Other laws may also apply. This is a
    list for Legal, not a rating.
[info] HR_TERMS that matched no register row: hiring, candidate, promotion, performance review,
    termination, employee monitoring

== GOVERN 1.5 / MANAGE 4.1: recording_daily
period      : 2026-07-06 to 2026-09-27 (UTC, inclusive)
filters     : one row per day; records counted by created_at date; enabled on 2026-06-01
run at      : 2026-10-31 16:45:27 UTC
rows        : 84
sha256      : 8b36c1c7e029fb6a7b1e70be1d3f9a3a94575bd3f9ace779ae243f783b8cd54d
    days recorded: 84 of 84
    from 2026-06-01 to the period's end, days with no records: 0

== Result
input checks : all passed
follow-up    : 0 item(s) need an explanation (incident rows and register rows with a gap, days with
no records or before enablement)
RESULT: NO GAPS FOUND

```

Run 3: the enablement day not set

Command: `python3 nist_ai_rmfm_claude_evidence.py --enablement-unknown`

```

input      : GENERATED records (not a real organization), seed 224, mode=clean
enabled on: not set (COMPLIANCE_API_ENABLED_ON is None)

== NIST documents (pinned: a different size or hash means NIST re-published the file)
  NIST AI 600-1 (Generative AI Profile, July 2024): 1174643 bytes, Last-Modified Mon, 24 Mar 2025
  19:11:27 GMT, sha256 6e73620ab6b64e90ef2c04bf0e0d6246185a2f4b1b13cab0df494496cff89b6a
  NIST AI 100-1 (AI RMF 1.0, January 2023): 1946127 bytes, Last-Modified Wed, 04 Jun 2025 17:14:26
  GMT, sha256 7576edb531d9848825814ee88e28b1795d3a84b435b4b797d3670eafdc4a89f1

== Input checks (activity records read: 694; before the period: 199; after the period's end: 16,
not used; incident log rows: 3; register rows: 6)
[ok ] NIST documents that are not the pinned file: 0
[ok ] duplicate activity ids: 0
[ok ] activity records with no id or no readable created_at: 0

```

```

[FAIL] the day the Compliance API was turned on is not set, so the pull cannot be checked: 1
(trips on: a pull that starts at the period; the enablement day not set)
[ok ] days with no records, from enablement (or the period's start) to its end: 0
[ok ] days in the period before the Compliance API was enabled: 0
[ok ] incident log rows repeated (same System ID, Title and Date of Incident): 0
[ok ] register ids listed more than once: 0
[ok ] register rows with no use case id: 0
[ok ] incident log and register dates not written YYYY-MM-DD: 0
      earliest record: 2026-06-01; enabled on: not set
[info] after the period's end, not checked: duplicate activity ids 1; records with no id 0
[info] incidents with a Date of Incident inside the period: 3 of 3 (0 with no Date of Incident
written YYYY-MM-DD, so not placed)

== GOVERN 4.3 / GV-4.3-002: incident_log_check  ** INCOMPLETE: see input checks **
period   : 2026-07-06 to 2026-09-27 (UTC, inclusive)
filters  : every row of the incident log, none dropped; row_no is the row's place in the file
after the header; System ID matched to the register on the lower-cased, trimmed id; the nine fields
as NIST's AI 600-1 lists them
run at   : 2026-10-31 16:45:33 UTC
rows     : 3
sha256   : d0e76435a95387a55368e26ebff452cc3eb92f861689db562fd296c18b376a81
  1 | UC-001 | 2026-07-18 | complete
  2 | UC-003 | 2026-08-11 | complete
  3 | UC-002 | 2026-09-02 | complete

== MAP 1.1 / MP-1.1-002: use_case_register_check  ** INCOMPLETE: see input checks **
period   : 2026-07-06 to 2026-09-27 (UTC, inclusive)
filters  : every register row, none dropped; first applicable status; review overdue means last
reviewed before 2026-09-27 minus 180 days; FOR LEGAL TO CHECK marks HR and people use cases by your
HR_TERMS and HR_DEPARTMENTS
run at   : 2026-10-31 16:45:33 UTC
rows     : 6
sha256   : b9e339a468c693d39c642bde8faad2bd673d71014453bc4f1cb315653ed35b69
  1 | UC-001 | 2026-06-15 | complete | -
  2 | UC-002 | 2026-07-02 | complete | -
  3 | UC-003 | 2026-08-20 | complete | -
  4 | UC-004 | 2026-05-28 | complete | -
  5 | UC-005 | 2026-09-01 | complete | FOR LEGAL TO CHECK
  6 | UC-006 | 2026-07-30 | complete | -
    for Legal to check: 1 use case(s). Reason: laws on automated employment decisions, such as NYC
Local Law 144, Colorado SB 26-189 (bill text not verified) and California's automated
decisionmaking technology rules under the CCPA, may apply. Other laws may also apply. This is a
list for Legal, not a rating.
[info] HR_TERMS that matched no register row: hiring, candidate, promotion, performance review,
termination, employee monitoring

== GOVERN 1.5 / MANAGE 4.1: recording_daily  ** INCOMPLETE: see input checks **
period   : 2026-07-06 to 2026-09-27 (UTC, inclusive)
filters  : one row per day; records counted by created_at date; enabled on not set
run at   : 2026-10-31 16:45:33 UTC
rows     : 84
sha256   : 8b36c1c7e029fb6a7b1e70be1d3f9a3a94575bd3f9ace779ae243f783b8cd54d
  days recorded: 84 of 84
  from 2026-07-06 to the period's end, days with no records: 0

== Result
input checks : FAILED, the lists above are not complete
follow-up    : 0 item(s) need an explanation (incident rows and register rows with a gap, days with

```

```
no records or before enablement)
RESULT: GAPS FOUND
```

How to read the results

- **Input checks come first.** A `[FAIL]` line means the lists below it are printed with `INCOMPLETE` in their headings. Fix the input, then run again.
- **Incident log check.** `MISSING` names the NIST fields that are blank for that row. `NOT WRITTEN YYYY-MM-DD` names a date the script would not guess at. `SYSTEM NOT ON REGISTER` means either the register is out of date, the incident names the wrong system, or the id was mistyped. A row that reads `complete` has all nine fields, readable dates in order and a System ID on the register.
- **Use-case register check.** The first status that applies is shown. `complete` says the fields are filled and the review is recent, not that the context of use is good.
- **FOR LEGAL TO CHECK** is a list for your lawyers, shown whatever the row's other status. It is not a rating.
- **Recording by day.** `NO RECORDS` is a day with none; `BEFORE ENABLED` is a day before the Compliance API was turned on, which Anthropic does not backfill.
- **RESULT.** `NO GAPS FOUND` says the records are filled in, with readable dates. `GAPS FOUND` says an input check failed, or that some rows or days need an explanation. With `--strict` the script exits with code 1 when there are gaps.
- **Fingerprints.** Each list's SHA-256 lets anyone who re-runs the script on the same files confirm they got the same rows.

Checklist

Done	Step
☐	Pull the Activity Feed from the day the Compliance API was turned on, and set <code>COMPLIANCE_API_ENABLED_ON</code> to that day.
☐	Export the incident log as a CSV with a column for each of NIST's nine fields, and put each column's name in <code>INC_COLS</code> .
☐	Export the use-case register as a CSV and put each column's name in <code>UC_COLS</code> . Use the same id for a use case in both files; the script ignores case and spaces around it.
☐	Write every date <code>YYYY-MM-DD</code> .

Done	Step
☐	Set PERIOD_START, PERIOD_END, REVIEW_AS_OF and REVIEW_MAX_DAYS to your own period and policy.
☐	Run the script and read the input checks first. If one fails, fix the input before you use the lists.
☐	For each incident row that is not complete, complete the record, or write down why it cannot be.
☐	For each register row that is not complete, document its context of use, owner or review.
☐	Send the use cases marked FOR LEGAL TO CHECK to your lawyers.
☐	For each day marked NO RECORDS, write down the explanation; list explained days before the period in KNOWN_GAP_DAYS.
☐	Keep each list's CSV and its printed fingerprint with the date you ran the script.
☐	Write down who reads the Activity Feed and how often. The script cannot do this for you.
☐	Ask Anthropic for the third-party documents your GOVERN 6.1 policy names.

Limits

- **This is not a client result.** The records are generated. The runs show the script finds planted faults. They say nothing about any real organization or how often such faults happen.
- **We did not call the Compliance API,** and we have not seen a real record.
- **Following NIST is not something a script can show.** The framework is voluntary and gives no pass mark. The script checks that the incident log holds the nine example fields NIST lists, that each register row states a context of use, and that the feed recorded each day. Owner, risk tier and review date are our checks.
- **The Activity Feed fields the script reads carry no incident report and no use-case id.** Both lists depend on records your company keeps, and the script cannot tell whether the incident log or the register is complete.
- **Coverage:** The Compliance API covers Claude Enterprise. For Claude Console and API-key workloads, the Activity Feed records administrative and resource events but no prompt or response text. Claude Code sessions authenticated with a Console API key, run through Amazon Bedrock, Google Cloud or Microsoft Foundry, or run as Claude Code cloud sessions are not captured ([Anthropic: Compliance API FAQ](#) (read 2026-10-31)). Recording starts when the Compliance API is first turned on.
- **A recorded day is not a reviewed day.** GOVERN 1.5 and MANAGE 4.1 ask for monitoring that is planned and carried out. The script shows the record exists, not that anyone read it.

- **The HR mark is a list for Legal.** Whether any of these laws applies to a use case is a question for your lawyers.
- **Our reading is not NIST's.** The mapping of ids to lists is ours. NIST has not reviewed it.
- **The generated incidents and use cases are invented.** Names, ids and addresses in them belong to no one.
- **Your incident log may hold workforce emails and sensitive detail,** so store it under the access controls you use for other security records. The script runs on your machine and sends nothing to Data Meaning. It does fetch the two NIST PDFs.

Sources

- [NIST AI 600-1, Generative AI Profile \(July 2024\)](#) (read 2026-10-31)
- [NIST AI 100-1, AI RMF 1.0](#) (read 2026-10-31)
- [NIST: AI RMF Core](#) (read 2026-10-31)
- [Anthropic: Query the Activity Feed](#) (read 2026-10-31)
- [Anthropic: Compliance API FAQ](#) (read 2026-10-31)
- [Anthropic: Delivery guarantees and completeness](#) (read 2026-10-31)
- [Anthropic Trust Center](#) (read 2026-10-31)
- [DuckDB: keywords and identifiers](#) (read 2026-10-31)
- [DuckDB: CSV overview](#) (read 2026-10-31)
- [DuckDB: date format functions](#) (read 2026-10-31)
- [NYC DCWP: Automated Employment Decision Tools](#) (read 2026-10-31)
- [California Privacy Protection Agency: regulations](#) (read 2026-10-31)

See Control Tower for Claude

Control Tower for Claude runs in your own Azure subscription.

www.datameaning.com/solutions/claude-control-tower.html