MCP HARDENING RECORD

USASpending.gov MCP Testing Record

MCP server hardened across five release cycles and four audit rounds against the live production API

Skill
usaspending-gov-mcp v0.2.3
github.com/1102tools/usaspending-gov-mcp

Date
April 2026
1102tools.com

Executive Summary

This Model Context Protocol server exposes the USASpending.gov REST API as 17 callable tools for federal contract and award research. It was hardened across five release cycles and four audit rounds, including a deep live audit against the production USASpending.gov API. Testing surfaced 15 priority issues plus more than 28 additional integration bugs, all of which were fixed before the current release. The MCP ships with 62 regression tests that run on every change and a live-gated suite that can be executed against the real API on demand.

Metric	Value
MCP tools exposed	17
Total regression tests	62 (52 offline, 10 live-gated)
Audit rounds completed	4
Initial integration issues (round 1)	28+
P1 silent-wrong-data bugs found and fixed	10
P2 validation gaps found and fixed	5
Release cycles	5 (v0.1.2 through v0.2.3)
Current release	0.2.3
PyPI status	Published as `usaspending-gov-mcp`, auto-publishes via Trusted Publisher on tag push

What Was Tested

The MCP exposes 17 tools spanning the USASpending.gov API surface. Testing covered all of them end-to-end.

Search and aggregation: `search_awards`, `get_award_count`, `spending_over_time`, `spending_by_category`

Award detail: `get_award_detail`, `get_transactions`, `get_award_funding`, `get_idv_children`

Workflow convenience: `lookup_piid` (auto-detects contract vs IDV)

Autocomplete: `autocomplete_psc`, `autocomplete_naics`

Reference: `list_toptier_agencies`, `get_agency_overview`, `get_agency_awards`, `get_naics_details`, `get_psc_filter_tree`, `get_state_profile`

Each tool was exercised for argument validation, input sanitization, response-shape guarantees, error translation, pagination edge cases, and real-world data handling against the live production API.

How It Was Tested

Testing discipline

Prior unit tests in v0.1.x awaited raw coroutines directly, which bypassed the FastMCP tool pipeline and its pydantic validation layer. This skipped whole categories of bugs. The hardening program switched to invoking tools through `mcp.call_tool(name, kwargs)` the way a real MCP client does. That change alone surfaced more than 28 integration issues invisible to the prior test suite.

Audit rounds

Round	Scope	Probe count	Finding class
1	Integration stress through real MCP client	83 live probes across all 17 tools	28+ integration issues
2	Targeted live probes on edge cases (null bytes, negative amounts, empty-string arrays, whitespace IDs, retired NAICS codes, reversed date ranges)	49 probes	9 P1 silent-wrong-data, 4 P2 validation
3	Deep live stress (compound filters, pagination boundaries at page 200 and 201, leap-year dates, 10-year spans, amount boundaries, unicode, agency name variations, 5 concurrent calls)	52 probes	1 additional P1: `search_awards()` with no filter arguments silently returned 25 unfiltered recent contracts
4	Response-shape mock fuzzing (None, bare list, int, string where a dict was expected)	15 probes	Response-shape guard gap

Live audit status

All four rounds included live calls against the production USASpending.gov API. The repository includes 10 live-gated regression tests executable via `USASPENDING_LIVE_TESTS=1 pytest` covering real search with real results, compound filters, leap-year dates, exact-match amount ranges, autocomplete returns, state profile, concurrent searches, unicode keyword handling, and toptier-agency listing.

Issues Found and Fixed

Priority 1: Silent wrong-data bugs

These are the most dangerous class: the tool returned data, but the data was wrong or unfiltered in a way the caller could not detect. All ten were found across rounds 1 through 3 and fixed in v0.2.0, v0.2.1, and v0.2.2.

Issue	Fix
`search_awards()` with no filter arguments silently returned 25 unfiltered recent contracts (same failure-mode category as regulations.gov-mcp's `agency_id=""` returning all 1.95 million records)	Raises "at least one filter beyond award_type" with pointer to typical filter combinations
Null byte, newline, tab in `keywords` silently accepted or produced upstream 500s	All free-text fields reject control characters up front
Null byte in autocomplete `search_text` produced upstream 500s	Rejected locally before HTTP call
Null byte in `generated_award_id` / `generated_idv_id` produced upstream 500s	Rejected locally on all detail tools
Negative `award_amount_min` / `award_amount_max` silently ignored by USASpending, returning default 25 results	Rejected with explanatory error
Lists of empty strings (`naics_codes=[""]`, `psc_codes=[""]`, `award_ids=[""]`) silently dropped to empty, applying no filter	Rejected with "contains only empty / whitespace strings" error
Empty or whitespace-only `generated_award_id` round-tripped to cryptic 422 or 404	Rejected up front with pointer to `search_awards` for valid IDs
Pydantic `extra='ignore'` default let typos like `keyword='cyber'` (real param is `search_text`) silently drop the typo'd argument and return unfiltered results	Every tool now applies `extra='forbid'` to its pydantic arg model; typos raise "Extra inputs are not permitted" before any HTTP call
Empty filters on `get_award_count` and `spending_over_time` forwarded to the API which then 400'd	Raises `ValueError` locally with filter guidance
Short autocomplete queries returned arbitrary first-N alphabetical records (e.g. "R" returning 10 unrelated GUN PSCs, "x" matching substring inside "(except potato)")	Minimum 2-character query enforced; retired NAICS codes filtered by default via `exclude_retired=True`

Priority 2: Validation gaps

Issue	Fix
`limit` unbounded on search, autocomplete, and convenience tools	Bounded to API caps (100 for search endpoints, 5000 for transactions)
`page` parameter unbounded (accepted 0, negative)	Required `>= 1` across all paginated tools
Date parameters accepted ISO 8601 datetimes, slash-separated, reversed ranges	Validated as `YYYY-MM-DD`, reversed ranges raise actionable error
`award_amount_min > award_amount_max` silently returned zero results	Raises with clear error message
`autocomplete_psc` and `autocomplete_naics` long queries triggered upstream 500s	Capped at 200 characters

Response-shape defense

The `_post` and `_get` helpers now guarantee a dict return via `_ensure_dict_response`. USASpending always returns JSON objects for the endpoints this MCP uses. Anything else (None, bare list, int, string) is a CDN or proxy issue that previously leaked into tool output as a type confusion error. It now surfaces clearly as "USASpending returned an empty body at {path}" or "unexpected {type} at {path}".

Test Coverage

The repo ships 62 regression tests across four files. All 62 pass on every release cycle.

File	Purpose	Test count
`tests/test_validation.py`	Main regression suite covering every round-1 through round-4 finding, plus 10 live-gated integration tests	62
`tests/stress_test.py`	Round 1 stress test scenarios (retained for reproducibility)	N/A (scenario script)
`tests/stress_test_r2.py`	Round 2 live-audit scenarios (retained for reproducibility)	N/A (scenario script)
`tests/stress_test_r3.py`	Round 3 deep live stress scenarios (retained for reproducibility)	N/A (scenario script)

Regression tests invoke tools through the FastMCP registry (`mcp.call_tool`) rather than awaiting decorated coroutines directly. This catches bugs in the tool pipeline that raw-coroutine tests miss. An autouse fixture resets `srv._client` between tests so the shared httpx client does not leak across event loops, preventing flaky test results from async state carryover.

Release History

Version	Focus	Regression test count
0.1.2	Initial release: 17 tools with basic unit tests	Basic coverage
0.2.0	Integration stress testing through real MCP client surfaced 28+ integration issues; added comprehensive input validation, bounds checking, and error hygiene	Expanded offline + integration suite
0.2.1	Cross-MCP fix discovered during sam-gov-mcp audit: pydantic `extra='forbid'` applied to all tool arg models to prevent typo'd-parameter silent filter-drop bugs	+1 regression test
0.2.2	Live audit surfaced 9 P1 silent-wrong-data paths and 4 P2 validation gaps; all fixed	46 total (+17 regressions)
0.2.3	Round 3 deep live stress and round 4 response-shape mock fuzz; added the `search_awards()` no-filter guard and `_ensure_dict_response` guarantee; live-gated regression suite	62 total (+16 regressions)

Cross-MCP Context

This MCP is one of eight servers in the 1102tools federal-contracting MCP suite (`bls-oews-mcp`, `ecfr-mcp`, `federal-register-mcp`, `gsa-calc-mcp`, `gsa-perdiem-mcp`, `regulationsgov-mcp`, `sam-gov-mcp`, and this one). All eight were hardened under the same playbook. Several fixes here originated in another MCP's audit and propagated across the suite:

- **`extra='forbid'` on pydantic arg models** was discovered during the sam-gov-mcp 0.3.1 audit after a typo'd parameter silently returned an unfiltered default. Applied here in 0.2.1 and to every other MCP in the suite.
- **No-filter guard on search tools** (the `search_awards()` fix) used the same pattern as the regulationsgov-mcp fix for `agency_id=""` returning all 1.95 million records. Same failure mode, same fix shape.
- **Response-shape guarantees** via `_ensure_dict_response` use the same defensive-parsing pattern applied across gsa-perdiem-mcp, bls-oews-mcp, and others where upstream APIs occasionally return non-JSON or shape-shifted responses.

What Was Not Tested

- **Rate-limit behavior.** USASpending does not document rate limits publicly. The MCP passes through whatever limits the API enforces but does not implement client-side throttling. Heavy concurrent use may hit limits the MCP cannot anticipate.
- **Historical API changes.** Tests validate behavior against the current USASpending API. Breaking changes to the upstream API (field renames, endpoint deprecations) are not caught by offline tests. Live-gated tests will catch them but must be run manually with `USASPENDING_LIVE_TESTS=1`.
- **Payload size limits beyond `limit` capping.** Response sizes over ~95KB are theoretically possible on some endpoints if the caller accepts the default shape. The MCP does not enforce an overall payload size ceiling.
- **Pending API deprecation.** USASpending has signaled that `subawards` award type will be superseded by a `spending_level` parameter. The MCP does not yet expose `spending_level`. When upstream fully deprecates, grants queries may need an adjustment.

Verification

All testing artifacts are in the repository. The methodology and fixes are reviewable commit-by-commit in git history. The regression test suite runs via `pytest` in the repo root and can be re-executed by anyone. The live suite runs with `USASPENDING_LIVE_TESTS=1 pytest` and requires no API key (USASpending is a free, public API).

> **Testing Methodology**
>
> Evaluators: James Jenrette, 1102tools, with Claude Code Opus 4.7 (1M context, max effort, Claude Max 20x subscription) during the hardening playbook execution.
>
> Testing spanned four rounds from integration stress testing through live API audits and response-shape guards. The live regression suite runs against the USASpending.gov production API when enabled with `USASPENDING_LIVE_TESTS=1`.
>
> Test count: 62 regression tests. P1 bugs found and fixed: 10. P2 validation gaps closed: 5. Integration issues closed in round 1: 28+. Release cycles: 5. Current version: 0.2.3. PyPI: `usaspending-gov-mcp`.
>
> Source: github.com/1102tools/usaspending-gov-mcp. License: MIT.