Notes with users — the security report

Produced by `sv` 0.3.0 (commit 7cb7878b2ec7) on 2026-11-22 03:24:15 UTC, run 90bae36f9345. ASVS level 2.

This file: The whole report as one file to read, print, or send: compliance.md's order, with each finding in full. US Letter paper unless stackvet.toml's report-paper setting asks for A4; its fonts show Western European letters only, so report.html is the complete version. **For:** You, or whoever you hand it to. The end of `report.html` lists every file in this folder and who each is for.

This is the report as one file to read or print. The fonts every PDF reader has show Western European letters only, so any other character in it prints as ?. report.html, beside it, is the complete version.

The short version

2 things to fix were found in the app's own code. 1 more is a note worth knowing, not a thing to fix, under "Worth knowing".

Most serious first:

- **A password is stored, or a key made from it, with too few rounds of work** — medium (ast.weak-password-key-derivation)
- **There is no way to report a security problem** — low (config.security-contact)

Nothing shows that sv was used while this app was built: no call from an AI coding tool to sv's MCP server is recorded for it. That is not a finding. The app may have been checked at a terminal, or built before sv kept this record.

This run took 0.7 s. The slowest parts: Putting the report together (0.3 s), Reading its configuration (0.2 s), Reading its code (0.1 s), Looking for keys and passwords (0.0 s), Recognizing its languages and frameworks (0.0 s).

The app's software bill of materials is in sbom.cdx.json beside this report. It lists no packages: the app declares none, in a package list or a lockfile.

stackvet.sarif beside this report is for code editors and GitHub's code scanning, not for reading: it holds 3 findings, and 10 notes on what was not examined.

Of the requirements that apply to this app:

- **2** — something found a problem
- **1** — an automated check looked and found nothing wrong
- **7** — an automated check tried part of what each asks and found nothing wrong there
- **185** — nothing has looked at these at all

Held to ASVS level 2. Not in these numbers: 153 more requirements at level 3, and 241 not yet placed because nobody has answered the question that decides them. Level 2 because stackvet.toml says customers use it: answers in stackvet.toml, which your AI coding tool usually writes and nobody has confirmed, so check them, and confirm them with `sv review`.

Not run this time: the running app, signed in or not (`sv report --run`), and outside tools such as Semgrep (they run with `sv report` at a terminal; the MCP server never runs them). 59 of the requirements that apply can only be checked that way.

What to do next

1. Fix the 2 things that were found. *(each is below, and in security.md with what to do)*
2. Answer the 23 questions at level 1 that no tool can settle — what your rules are, who may do what, how the app is built. 69 in all, most of them design review at higher levels. *(ask your AI coding tool to ask you them one at a time (it calls stackvet_check with section "questions" and records your answers); at a terminal,* `sv questions` *prints them)*
3. Ask your AI coding tool to write tests for the 116 requirements a test could settle, starting with the 43 at level 1. *("Tests to write", lowest level first)*
4. Run the full report, which starts the app behind sv's network fence and runs the outside tools: 59 requirements can only be checked that way. *(*`sv report` *does not start the app unless you pass --run.)*
5. Answer the questions in stackvet.toml that would place 241 more requirements, which are in none of the numbers above until then. *("Requirements nobody has placed" names the question for each)*

2 things to fix

Worst first.

[medium] A password is stored, or a key made from it, with too few rounds of work

Where: `app.py` line 49

How sure: confirmed.

Fingerprint: `v2-9d2dd4b16e6a08fd`*. A person who has looked and found it a false alarm, or a risk to live with for now, can set it aside under* `[[finding-review]]` *in stackvet.toml.*

Next step. Fix it in the code and check again. If it is a false alarm, write a `[[finding-review]]` entry for it, naming its rule, file and fingerprint, and confirm it with `sv review` in your own terminal; until then it counts.

A password is hashed for storage, or a key derived from it, with PBKDF2, and the number of rounds written into the code is below what OWASP recommends: 600,000 for SHA-256 and 210,000 for SHA-512 (SHA-1 needs 1,300,000). A count below 210,000 is reported whatever the hash; one below 600,000 where the call itself names SHA-256, and one below 1,300,000 where it names SHA-1.

The code.

```
    48  def hash_password(password, salt):
>   49      return hashlib.pbkdf2_hmac("sha256", password.encode(), salt.encode(), 200_000).hex()
    50
```

Why it matters. Anyone who gets hold of what the key protects, such as an encrypted backup, an export, or a stored hash, can try guesses at the password quickly on ordinary hardware, and most people's passwords fall to a fast guesser.

What to do. Raise the number of rounds to at least 600,000 for PBKDF2 with SHA-256 (210,000 with SHA-512, and 1,300,000 with SHA-1, which is better replaced), or use a password hashing function made for this, such as Argon2id or scrypt. Keep the count where it can be raised later without breaking what was already stored.

Evidence about: V11.4.2, V11.4.4

Known as: CWE-916

1 low-severity finding, by rule

Each is in full, with where and how to fix it, under "Every low-severity finding", near the end.

- `config.security-contact` — 1 time, in `SECURITY.md`

Worth knowing: 1 note

Not things to fix. Each says something about how the app is built that is worth knowing, and asks for nothing unless you decide it matters; if you do, its "What to do" says how.

[info] 138 commit(s) on the default branch were made by an AI coding tool and not merged in

Where: `.git` line 1

How sure: confirmed.

Fingerprint: `4213cb48a86e8e56`*. A person who has looked and found it a false alarm, or a risk to live with for now, can set it aside under* `[[finding-review]]` *in stackvet.toml.*

These commits name an AI coding tool and reached the branch directly: `1fe46225` Run the app's container read-only, without capabilities (#496) (Claude Opus 5.5 <noreply@anthropic.com>); `93bb7454` Claim running the app's container read-only (#493) (Claude Opus 5.5 <noreply@anthropic.com>); `53154bca` Backlog: say when a manifest and its lockfile disagree (#484) (Claude Opus 5.5 <noreply@anthropic.com>); `67a497af` Example app: PyJWT 2.15.1, and a lock that matches its manifest (#482) (Claude Opus 5.5 <noreply@anthropic.com>); `aeb056e1` Backlog: a prompt library from the CSA guide and lessons learned (#476) (Claude Opus 5.5 <noreply@anthropic.com>); `e91c7173` Backlog: record what the sessions agreed about the promotion (#221) (Claude Opus 5.5 <noreply@anthropic.com>); `8ced5fb1` Backlog: v1-paper is tagged and released (#216) (Claude Opus 5.5 <noreply@anthropic.com>); `76156b3c` Backlog: promote sv to the top of the repository, keep v1 for the paper (#210) (Claude Opus 5.5 <noreply@anthropic.com>); `0acfe495` Backlog: what the owner's first build from scratch found, and a zip (#204) (Claude Opus 5.5 <noreply@anthropic.com>); `ce032691` Backlog: packaging sv, a container now and a download later (#198) (Claude Opus 5.5 <noreply@anthropic.com>); and 128 more. This check reads only what a commit says about itself, in its `Assisted-by:` or `Co-Authored-By:` line, so a commit that names no tool is not seen. Whether the hosting service's protection rules held is in its settings, which `sv` does not read.

Why it matters. A change that reached the branch without a merge reached it without another person's review.

What to do. Ask the AI coding tool to work on a branch and open a pull request, and merge it after a look. Turn on the pull-request rule for the default branch, so a direct push is refused.

How this report was made

195 requirements apply to this app. Of those, **10 have been looked at by something** and **185 have not been looked at at all**.

A further 33 about how the app is built with an AI coding tool, from OWASP AISVS Appendix C, are counted apart: see "How the app is built with AI", below.

There is no line in this report that says a requirement passed, because nothing here is able to establish that. A requirement marked *checked* had at least one automated check look at it and find nothing wrong, over the coverage named beside it — which is worth having and is not the same as the requirement being met. One

marked as answered in the security notes, checked by hand, or answered yes rests on your word or your AI coding tool's, which is shown as exactly that and never as checked. Everything else applicable is *not verified*: nothing has produced evidence either way.

A further **241 requirements could not even be placed**: nobody has answered the question that decides whether they apply. They are listed at the end with the question in each case. They are not exclusions and they are not passes.

	count
Applies, needs attention	2
Applies, checked by an automated check	1
Applies, checked in part: an automated check tried some of what it asks	7
Applies, the app's own tests ran without failing: written by your AI coding tool, not a check of sv's	0
Applies, answered in the security notes: by you, or written by your AI coding tool and confirmed by a person	0
Applies, rests on your word: you checked it by hand	0
Applies, rests on your word: you answered yes about how it is built	0
Applies, rests on your AI coding tool's word: it answered yes	0
Applies, not verified by anything	185
Does not apply	18
Not assessed — nobody has answered	241
Above this app's target level (ASVS level 2)	153

What was not examined

Each of these is a limit on what the rest of this report can mean.

not examined	why
the running app	`sv report` does not start the app unless you pass --run. Without it, nothing here has asked the app anything — what it sends to a browser, what it says when something goes wrong, which sites it accepts.
the security tool this language already has	`sv report` ran none of the outside tools, because --no-tools was given; without it they run. Bandit, gosec, Brakeman, and Semgrep each know their languages far better than the handful of rules built in here.
the check `config.gitignore-covers-env`	This app is a folder inside a larger git repository and has no .gitignore of its own. A .gitignore in a folder above it may leave out .env, but `sv` reads only the app's folder, so this check did not run.
the check `config.security-contact-placeholder`	there is no SECURITY.md or security.txt to read, which config.security-contact reports
the check `config.versions-pinned`	No package manifest was found, so there is nothing whose versions could be pinned. If this app installs dependencies some other way, that is not something `sv` can see.
the check `config.development-server-started`	There is no Dockerfile or Procfile (other than ones named for development), so `sv` cannot see what command starts the app where it is deployed.

not examined	why
42 requirements that are design review, not scanning	these ask how the system was designed and how it is run — whether trust zones are enforced, whether an incident response plan is rehearsed, whether data has named owners. No check here reaches them and none ever will, so they are counted as applicable and unverified, and a person has to answer them.
known vulnerabilities in the packages this app ships	`sv report` compares against known vulnerabilities only when you pass --advisories DIR. `sv` does not fetch anything, because the list of packages an app depends on is yours: download an OSV export for this app's ecosystems, unpack it, and pass its folder with --advisories.
18 requirements that ask for a written decision	No tool can answer these: they ask what your rules are, who may do what, and how long things are kept. Run `sv notes` to write security-notes.md, answer the questions in it, and they become documented. Your AI coding tool can ask you them: `sv questions` prints them for its chat.
14 questions about how this app is built	No tool can settle these — whether input is validated on the server, whether the app's own services authenticate to each other. Answer them in the `[design]` section of stackvet.toml: V2.2.2, V7.2.1, V8.3.1, V15.3.1, V1.1.1, V4.1.3, V4.2.1, V8.2.3, V11.2.2, V14.2.2, V15.3.4, V16.2.2, V16.4.3, V2.3.3. Your AI coding tool can ask you them: `sv questions` prints them for its chat.

Outside tools

None ran this time: Bandit, gosec, Brakeman, Semgrep. They run with every `sv report` at a terminal unless `--no-tools` is given; the MCP server and the GitHub Action never run them.

Requirements that apply

Each chapter of the standards, with how many of its requirements apply to this app and what is known about them, and how many do not apply. What each requirement asks for is in the appendix at the end.

chapter	apply	a problem found	checked	checked in part	not verified	does not apply	not placed yet
V1 Encoding and Sanitization	**15**	0	1	3	11	9	3
V2 Validation and Business Logic	**11**	0	0	0	11	0	0
V3 Web Frontend Security	**17**	0	0	1	16	1	1
V4 API and Web Service	**4**	0	0	0	4	6	0
V5 File Handling	**9**	0	0	1	8	0	0
V6 Authentication	**29**	0	0	0	29	1	5
V7 Session Management	**15**	0	0	0	15	0	3
V8 Authorization	**6**	0	0	0	6	0	1
V9 Self-contained Tokens	**0**	0	0	0	0	0	7
V10 OAuth and OIDC	**0**	0	0	0	0	0	29
V11 Cryptography	**14**	2	0	2	10	0	0
V12 Secure Communication	**4**	0	0	0	4	1	4
V13 Configuration	**9**	0	0	0	9	0	4
V14 Data Protection	**9**	0	0	0	9	0	0

chapter	apply	a problem found	checked	checked in part	not verified	does not apply	not placed yet
V15 Secure Coding and Architecture	**12**	0	0	0	12	0	1
V16 Security Logging and Error Handling	**15**	0	0	0	15	0	1
V17 WebRTC	**0**	0	0	0	0	0	7
SBD-AC Access Control & Secure Communication	**5**	0	0	0	5	0	2
SBD-AS Architecture & Service Design	**1**	0	0	0	1	0	7
SBD-DM Data Management & Protection	**3**	0	0	0	3	0	3
SBD-MT Monitoring, Testing & Incident Readiness	**7**	0	0	0	7	0	0
SBD-RR Reliability & Resilience	**5**	0	0	0	5	0	3
C1 Training Data Integrity & Traceability	**0**	0	0	0	0	0	11
C2 Input Validation	**0**	0	0	0	0	0	10
C3 Model Lifecycle Management & Change Control	**0**	0	0	0	0	0	11
C4 Infrastructure, Configuration & Deployment Security	**0**	0	0	0	0	0	5
C5 Access Control & Identity for AI Components & Users	**0**	0	0	0	0	0	6
C6 Supply Chain Security for Models	**0**	0	0	0	0	0	7
C7 Model Behavior, Output Control & Safety Assurance	**2**	0	0	0	2	0	8
C8 Memory, Embeddings & Vector Database Security	**0**	0	0	0	0	0	8
C9 Orchestration & Agentic Security	**0**	0	0	0	0	0	26
C10 Model Context Protocol (MCP) Security	**0**	0	0	0	0	0	21
C11 Adversarial Robustness	**0**	0	0	0	0	0	12
C12 Monitoring, Logging & Anomaly Detection	**3**	0	0	0	3	0	16
AISVS Appendix C: how the app is built with AI	**0**	0	0	0	0	0	19

Not in this table: the 33 requirements about how the app is built with an AI coding tool that nothing has reached, which are counted apart in "How the app is built with AI", below. The Appendix C row counts only the ones that have evidence, do not apply, or wait on a question.

V1 Encoding and Sanitization — 15 apply, 9 do not

requirement	status	level	in plain words

requirement	status	level	in plain words
V1.2.1	not verified — an outside tool could (they run with `sv report` at a terminal); a test of the app's could (ask your AI coding tool to write one)	1	Text the app puts into a page, a header, or an XML document is encoded for the exact place it goes, so it cannot change the page's structure.
V1.2.2	not verified — a test of the app's could (ask your AI coding tool to write one)	1	Untrusted data put into a web address is encoded for where it goes in the address, and only safe kinds of address (never javascript: or data:) are allowed.
V1.2.3	not verified — a test of the app's could (ask your AI coding tool to write one)	1	Untrusted data put into JavaScript or JSON the app builds is encoded or escaped, so it cannot change the script's or the JSON's structure.
V1.2.4	checked (a database query, sent through the usual database libraries' query calls, joined together from text and values rather than sent with its values kept separate, in 2 python files (the calls it reads: `execute`, `executemany`, `executescript`, `raw`, `read_sql`, `read_sql_query`, `extra`, `RawSQL`, `$where`, `or_`, and `filter`) (ast.sql-built-by-hand))	1	Database queries keep the data apart from the query itself (parameterized queries, an ORM, or something as safe), so nobody can inject their own query.
V1.2.5	checked in part (a command line built from a value and handed to a shell to run, in 2 python files (the calls it reads: `system`, `popen`, `getoutput`, and `getstatusoutput`) (ast.shell-command))	1	The app never lets input become part of an operating-system command: commands are given their arguments separately, or the input is encoded for the command line.
V1.3.2	checked in part (code built from a value and handed to something that runs it, such as eval, in 2 python files (the calls it reads: `eval`, `exec`, and `compile`) (ast.dynamic-code-execution))	1	The app does not run text as code (eval, SpEL, and the like); where there is truly no alternative, any user input in it is sanitized first.
V1.1.1	not verified — a test of the app's could (ask your AI coding tool to write one)	2	Input is decoded into its plain form once, only when encoded data is expected, and before it is validated or cleaned, never after.
V1.1.2	not verified — a test of the app's could (ask your AI coding tool to write one)	2	Output encoding and escaping happen as the last step before the data is used by its interpreter, or by the interpreter itself.
V1.2.9	not verified — a test of the app's could (ask your AI coding tool to write one)	2	Special characters put into a regular expression are escaped, so they are not read as metacharacters.
V1.3.3	not verified — a test of the app's could (ask your AI coding tool to write one)	2	Data passed to a potentially dangerous place is sanitized first: only characters safe there are allowed, and input that is too long is trimmed.
V1.3.4	not verified — a check of the running app could settle it (`sv report --run`); a test of the app's could (ask your AI coding tool to write one)	2	SVG images people upload are validated or sanitized to keep only safe tags and attributes, with no scripts or foreignObject.

requirement	status	level	in plain words
V1.3.6	not verified — an outside tool could (they run with `sv report` at a terminal); a test of the app's could (ask your AI coding tool to write one)	2	When the app calls another service with untrusted data, the protocol, domain, path, and port are checked against an allowlist, so it cannot be steered elsewhere (server-side request forgery, SSRF).
V1.3.7	not verified — an outside tool could (they run with `sv report` at a terminal); a test of the app's could (ask your AI coding tool to write one)	2	Templates are never built from untrusted input; where there is no alternative, the input is sanitized or strictly validated, to stop template injection.
V1.3.10	not verified — a test of the app's could (ask your AI coding tool to write one)	2	Format strings that could resolve in an unexpected or malicious way are sanitized before they are processed.
V1.5.2	checked in part (data read with a reader that can build any kind of object, such as pickle, Marshal, or PHP's unserialize, in 2 python files (the calls it reads: `load`, `loads`, and `Unpickler`) (ast.unsafe-deserialization))	2	Deserialization of untrusted data is safe, for example with an allowlist of object types, and deserialization known to be insecure is never used on it.

V2 Validation and Business Logic — 11 apply

requirement	status	level	in plain words
V2.1.1	not verified — only you can (it is in "What only you can check")	1	The app's documentation says what valid data looks like for each kind of input: an email address, a phone number, a card number, or the app's own formats.
V2.2.1	not verified — a check of the running app could settle it (`sv report --run`); a test of the app's could (ask your AI coding tool to write one)	1	Input is checked against what the app expects (allowed values, patterns, ranges, or structure); at level 1, at least the input used to make business or security decisions.
V2.2.2	not verified — a test of the app's could (ask your AI coding tool to write one)	1	Input is validated on the server, the trusted side; checks in the browser help people but are never relied on for security.
V2.3.1	not verified — only you can (it is in "What only you can check")	1	A multi-step process, such as a checkout, runs its steps in order for the same user, and no step can be skipped.
V2.1.2	not verified — only you can (it is in "What only you can check")	2	The documentation says how to check that related pieces of data make sense together, such as a suburb matching its ZIP code.
V2.1.3	not verified — only you can (it is in "What only you can check")	2	The documentation sets out the business logic limits and validations, per user and across the whole application.
V2.2.3	not verified — a test of the app's could (ask your AI coding tool to write one)	2	The app checks that combinations of related data items are reasonable, by the rules set out beforehand.
V2.3.2	not verified — a check of the running app could settle it (`sv report --run`)	2	The business logic limits the documentation sets out are enforced in the application, so they cannot be exploited.
V2.3.3	not verified — a test of the app's could (ask your AI coding tool to write one)	2	Business operations use transactions: each succeeds as a whole or is rolled back to the previous correct state.
V2.3.4	not verified — only you can (it is in "What only you can check")	2	Limited resources, such as theater seats or delivery slots, are locked while being taken, so they cannot be double-booked.

requirement	status	level	in plain words
V2.4.1	not verified — a check of the running app could settle it (`sv report --run`); a test of the app's could (ask your AI coding tool to write one)	2	Anti-automation controls stop excessive calls to the application's functions, which could leak data, fill it with garbage, use up quotas or costly resources, or deny service.

V3 Web Frontend Security — 17 apply, 1 do not

requirement	status	level	in plain words
V3.2.1	not verified — a check of the running app could settle it (`sv report --run`); a test of the app's could (ask your AI coding tool to write one)	1	The browser cannot be tricked into showing a response in the wrong context, such as an API response or an uploaded file opened directly as a page.
V3.2.2	not verified — a check of the running app could settle it (`sv report --run`); a test of the app's could (ask your AI coding tool to write one)	1	Content meant to be shown as text is put on the page with safe functions (such as textContent), so it can never run as HTML or JavaScript.
V3.3.1	not verified — an outside tool could (they run with `sv report` at a terminal); a test of the app's could (ask your AI coding tool to write one)	1	Cookies have the Secure attribute, so they are only sent over HTTPS, and their names use the __Host- or __Secure- prefix.
V3.4.1	not verified — only the live site can show it (`sv probe` once it is on the internet); a test of the app's could (ask your AI coding tool to write one)	1	Every response tells browsers to use only HTTPS for this site (the Strict-Transport-Security header), for at least a year.
V3.4.2	not verified — a check of the running app could settle it (`sv report --run`); an outside tool could (they run with `sv report` at a terminal); a test of the app's could (ask your AI coding tool to write one)	1	Cross-origin access (CORS) is allowed only for a fixed origin or origins on an allowlist; when any origin is allowed, the response holds nothing sensitive.
V3.5.1	not verified — a check of the running app could settle it (`sv report --run`); an outside tool could (they run with `sv report` at a terminal); a test of the app's could (ask your AI coding tool to write one)	1	Requests that do something sensitive are checked to come from the app itself (with an anti-forgery token or a special header), to stop cross-site request forgery (CSRF), unless the CORS preflight already does that.
V3.5.2	not verified — a check of the running app could settle it (`sv report --run`); a test of the app's could (ask your AI coding tool to write one)	1	Where the app relies on the browser's CORS preflight to block other sites, sensitive actions cannot be reached by a request that skips the preflight.
V3.5.3	not verified — a test of the app's could (ask your AI coding tool to write one)	1	Sensitive actions use POST, PUT, PATCH, or DELETE, never GET or another method meant to be safe, or the request's Sec-Fetch headers are checked strictly instead.
V3.3.2	not verified — a check of the running app could settle it (`sv report --run`); a test of the app's could (ask your AI coding tool to write one)	2	Each cookie's SameSite attribute is set to suit its purpose, to limit cross-site request forgery (CSRF) and clickjacking.
V3.3.3	not verified — only the live site can show it (`sv probe` once it is on the internet); a test of the app's could (ask your AI coding tool to write one)	2	Cookie names start with __Host- unless the cookie is meant to be shared with other hosts.
V3.3.4	not verified — a check of the running app could settle it (`sv report --run`); a test of the app's could (ask your AI coding tool to write one)	2	A cookie that scripts in the page have no need to read, such as a session token, has the HttpOnly attribute, and its value is only ever sent in the Set-Cookie header.

requirement	status	level	in plain words
V3.4.3	not verified — a check of the running app could settle it (`sv report --run`); a test of the app's could (ask your AI coding tool to write one)	2	Every response has a Content-Security-Policy header that only lets the browser load and run trusted content, at least with object-src 'none' and base-uri 'none', and an allowlist, nonces, or hashes.
V3.4.4	not verified — a check of the running app could settle it (`sv report --run`); a test of the app's could (ask your AI coding tool to write one)	2	Every response has the header X-Content-Type-Options: nosniff, so browsers never guess a response's type.
V3.4.5	not verified — a check of the running app could settle it (`sv report --run`); a test of the app's could (ask your AI coding tool to write one)	2	The app sets a referrer policy, so addresses with sensitive data in them are not leaked to other sites through the Referer header.
V3.4.6	not verified — a check of the running app could settle it (`sv report --run`); a test of the app's could (ask your AI coding tool to write one)	2	Every response's Content-Security-Policy has a frame-ancestors directive, so other sites cannot embed the app's pages unless that is intended.
V3.7.1	not verified — a test of the app's could (ask your AI coding tool to write one)	2	The app uses only client-side technologies that are still supported and secure, never Flash, ActiveX, Silverlight, Java applets, or the like.
V3.7.2	checked in part (a redirect to a destination built from a value, in 2 python files (the calls it reads: `redirect`, `HttpResponseRedirect`, `HttpResponsePermanentRedirect`, and `RedirectResponse`) (ast.open-redirect))	2	The app redirects people automatically to another site it does not control only when that destination is on an allowlist.

V4 API and Web Service — 4 apply, 6 do not

requirement	status	level	in plain words
V4.1.1	not verified — a check of the running app could settle it (`sv report --run`); a test of the app's could (ask your AI coding tool to write one)	1	Every response with a body says what kind of content it is (the Content-Type header) and matches it, including the character set, such as UTF-8.
V4.1.2	not verified — a test of the app's could (ask your AI coding tool to write one)	2	Only pages people open in a browser redirect from HTTP to HTTPS; other endpoints do not, so a client sending unencrypted requests is noticed.
V4.1.3	not verified — a test of the app's could (ask your AI coding tool to write one)	2	Header fields that a proxy or load balancer sets, such as X-Forwarded-For or X-Real-IP, cannot be overridden by the end user.
V4.2.1	not verified — only you can (it is in "What only you can check")	2	Every component, from load balancer to application server, decides where each HTTP message ends in the same correct way, to prevent HTTP request smuggling.

V5 File Handling — 9 apply

requirement	status	level	in plain words
V5.2.1	not verified — a check of the running app could settle it (`sv report --run`); a test of the app's could (ask your AI coding tool to write one)	1	The app accepts only files small enough to process without slowing down or being knocked over.

requirement	status	level	in plain words
V5.2.2	not verified — a check of the running app could settle it (`sv report --run`); a test of the app's could (ask your AI coding tool to write one)	1	An uploaded file's extension is one the app expects, and its contents really are that type (checking its first bytes, for example); at level 1, at least files used for business or security decisions.
V5.3.1	not verified — a check of the running app could settle it (`sv report --run`); a test of the app's could (ask your AI coding tool to write one)	1	Files that people upload into a public folder can never be run as code on the server when someone opens their address.
V5.3.2	checked in part (a file opened, written, or deleted at a path built from a value rather than written out, in 2 python files (the calls it reads: `open`, `send_file`, and `FileResponse`) (ast.file-path-from-value))	1	File paths are built from names the app makes or trusts, not names people send; if a user's file name must be used, it is strictly checked, so nobody can reach other files or servers.
V5.1.1	not verified — only you can (it is in "What only you can check")	2	The documentation sets, for each upload feature, the allowed file types and extensions, the largest size (unpacked too), and how files are made safe to download, including what happens to a malicious one.
V5.2.3	not verified — a check of the running app could settle it (`sv report --run`); a test of the app's could (ask your AI coding tool to write one)	2	Before unpacking a compressed file (zip, gz, docx, odt), the app checks its unpacked size and number of files against set limits.
V5.4.1	not verified — a check of the running app could settle it (`sv report --run`); a test of the app's could (ask your AI coding tool to write one)	2	File names people submit are validated or ignored, and the app sets the file name itself in the Content-Disposition header of the response.
V5.4.2	not verified — a check of the running app could settle it (`sv report --run`); a test of the app's could (ask your AI coding tool to write one)	2	File names the app serves, in headers or email attachments, are encoded or sanitized (as RFC 6266 describes), so they cannot inject anything.
V5.4.3	not verified — only you can (it is in "What only you can check")	2	Files from untrusted sources are scanned by an antivirus scanner, so known malicious files are not served.

V6 Authentication — 29 apply, 1 do not

requirement	status	level	in plain words
V6.1.1	not verified — only you can (it is in "What only you can check")	1	The documentation says how the app defends sign-in against password guessing and stolen-password lists (rate limiting and the like), how that is set up, and how it avoids locking real users out.
V6.2.1	not verified — a check of the running app could settle it (`sv report --run`); a test of the app's could (ask your AI coding tool to write one)	1	Passwords people choose are at least 8 characters long; at least 15 is strongly recommended.
V6.2.2	not verified — a check of the running app could settle it (`sv report --run`); a test of the app's could (ask your AI coding tool to write one)	1	People can change their password.
V6.2.3	not verified — a check of the running app could settle it (`sv report --run`); a test of the app's could (ask your AI coding tool to write one)	1	Changing a password asks for the current password as well as the new one.

requirement	status	level	in plain words
V6.2.4	not verified — a check of the running app could settle it (`sv report --run`); a test of the app's could (ask your AI coding tool to write one)	1	A new or changed password is checked against at least the 3,000 most common passwords that fit the app's rules, and refused if it is one of them.
V6.2.5	not verified — a check of the running app could settle it (`sv report --run`); a test of the app's could (ask your AI coding tool to write one)	1	Passwords can contain any characters, with no rules demanding upper case, numbers, or symbols.
V6.2.6	not verified — a check of the running app could settle it (`sv report --run`); a test of the app's could (ask your AI coding tool to write one)	1	Password fields hide what is typed (type=password); the app may let people show it for a moment.
V6.2.7	not verified — a test of the app's could (ask your AI coding tool to write one)	1	People can paste passwords and use the browser's or another password manager.
V6.2.8	not verified — a check of the running app could settle it (`sv report --run`); a test of the app's could (ask your AI coding tool to write one)	1	The password is checked exactly as typed, never cut short or changed to lower or upper case.
V6.3.1	not verified — a check of the running app could settle it (`sv report --run`)	1	The defenses against password guessing and stolen-password lists are in place, as the app's documentation says they are.
V6.3.2	not verified — a test of the app's could (ask your AI coding tool to write one)	1	There are no default accounts such as root, admin, or sa, or they are turned off.
V6.4.1	not verified — a test of the app's could (ask your AI coding tool to write one)	1	First passwords and activation codes the system makes are random, follow the password rules, expire soon or after first use, and cannot be kept as the long-term password.
V6.4.2	not verified — a test of the app's could (ask your AI coding tool to write one)	1	There are no password hints or secret questions.
V6.1.2	not verified — only you can (it is in "What only you can check")	2	A list of words specific to the app, such as the organization's and products' names, is documented, so they can be kept out of passwords.
V6.1.3	not verified — only you can (it is in "What only you can check")	2	If there are several ways to sign in, the documentation lists them all, with the security controls and strength each must enforce consistently.
V6.2.9	not verified — a check of the running app could settle it (`sv report --run`); a test of the app's could (ask your AI coding tool to write one)	2	Passwords of at least 64 characters are allowed.
V6.2.10	not verified — a test of the app's could (ask your AI coding tool to write one)	2	A password stays valid until it is found to be compromised or the user changes it; people are never made to change passwords on a schedule.
V6.2.11	not verified — a check of the running app could settle it (`sv report --run`)	2	The documented list of words specific to the app is used to stop easy-to-guess passwords being created.
V6.2.12	not verified — only you can (it is in "What only you can check")	2	Passwords chosen at registration or when changing a password are checked against a set of breached passwords.
V6.3.3	not verified — only you can (it is in "What only you can check")	2	Signing in to the application takes multi-factor authentication, or a combination of single-factor methods.

requirement	status	level	in plain words
V6.3.4	not verified — only you can (it is in "What only you can check")	2	If there are several ways to sign in, none is undocumented, and each enforces the same security controls and strength.
V6.4.3	not verified — a test of the app's could (ask your AI coding tool to write one)	2	Resetting a forgotten password is secure and never skips multi-factor authentication that is turned on.
V6.4.4	not verified — only you can (it is in "What only you can check")	2	When someone loses a multi-factor authentication factor, their identity is checked again as thoroughly as when they first enrolled.
V6.5.1	not verified — a check of the running app could settle it (`sv report --run`); a test of the app's could (ask your AI coding tool to write one)	2	Lookup secrets, codes sent out of band, and time-based one-time passwords (TOTPs) work only once.
V6.5.2	not verified — a test of the app's could (ask your AI coding tool to write one)	2	Lookup secrets with less than 112 bits of entropy are stored hashed with an approved password hashing algorithm and a 32-bit random salt.
V6.5.3	not verified — a test of the app's could (ask your AI coding tool to write one)	2	Lookup secrets, out-of-band codes, and time-based one-time password seeds come from a cryptographically secure random number generator (CSPRNG), so they cannot be predicted.
V6.5.4	not verified — a test of the app's could (ask your AI coding tool to write one)	2	Lookup secrets and out-of-band authentication codes have at least 20 bits of entropy (6 random digits is usually enough).
V6.5.5	not verified — a check of the running app could settle it (`sv report --run`); a test of the app's could (ask your AI coding tool to write one)	2	Out-of-band authentication codes expire within 10 minutes, and time-based one-time passwords (TOTP) within 30 seconds.
V6.6.3	not verified — a check of the running app could settle it (`sv report --run`); a test of the app's could (ask your AI coding tool to write one)	2	Out-of-band authentication codes are protected against brute force guessing by rate limiting.

V7 Session Management — 15 apply

requirement	status	level	in plain words
V7.2.1	not verified — a check of the running app could settle it (`sv report --run`); a test of the app's could (ask your AI coding tool to write one)	1	Session tokens are checked on the server, the trusted side, every time.
V7.2.2	not verified — a check of the running app could settle it (`sv report --run`); a test of the app's could (ask your AI coding tool to write one)	1	Sessions use tokens made fresh for each session, never a fixed API key or secret.
V7.2.3	not verified — a test of the app's could (ask your AI coding tool to write one)	1	Session tokens that point to data on the server are unique and made by a secure random generator, with at least 128 bits of randomness.
V7.2.4	not verified — a check of the running app could settle it (`sv report --run`); a test of the app's could (ask your AI coding tool to write one)	1	Signing in, or signing in again, gives a new session token and ends the old one.

requirement	status	level	in plain words
V7.4.1	not verified — a check of the running app could settle it (`sv report --run`); a test of the app's could (ask your AI coding tool to write one)	1	Once a session ends (signing out, or it expires), it cannot be used again; the server forgets it, or self-contained tokens are refused after that point.
V7.4.2	not verified — a check of the running app could settle it (`sv report --run`); a test of the app's could (ask your AI coding tool to write one)	1	When an account is disabled or deleted, all of its active sessions end.
V7.1.1	not verified — only you can (it is in "What only you can check")	2	The documentation sets the session inactivity timeout and the absolute maximum session lifetime, and justifies any difference from NIST SP 800-63B's re-authentication rules.
V7.1.2	not verified — only you can (it is in "What only you can check")	2	The documentation says how many sessions one account may have at once, and what happens when the maximum number is reached.
V7.3.1	not verified — a check of the running app could settle it (`sv report --run`)	2	An inactivity timeout makes people authenticate again, as the risk analysis and documented decisions set.
V7.3.2	not verified — a check of the running app could settle it (`sv report --run`)	2	An absolute maximum session lifetime makes people authenticate again, as the risk analysis and documented decisions set.
V7.4.3	not verified — a check of the running app could settle it (`sv report --run`); a test of the app's could (ask your AI coding tool to write one)	2	After a password or other authentication factor is changed or removed, the user is offered the choice to end all their other active sessions.
V7.4.4	not verified — a check of the running app could settle it (`sv report --run`); a test of the app's could (ask your AI coding tool to write one)	2	Every page that requires sign-in has an easy, visible way to log out.
V7.4.5	not verified — a check of the running app could settle it (`sv report --run`); a test of the app's could (ask your AI coding tool to write one)	2	Administrators can end the active sessions of one user, or of all users.
V7.5.1	not verified — a check of the running app could settle it (`sv report --run`); a test of the app's could (ask your AI coding tool to write one)	2	Changing anything that affects authentication or account recovery, such as the email address, phone number, or multi-factor settings, requires full re-authentication.
V7.5.2	not verified — a check of the running app could settle it (`sv report --run`); a test of the app's could (ask your AI coding tool to write one)	2	Users can see their active sessions and, after authenticating again, end any or all of them.

V8 Authorization — 6 apply

requirement	status	level	in plain words
V8.1.1	not verified — only you can (it is in "What only you can check")	1	The documentation sets out who may use which functions, and who may see or change which records.
V8.2.1	not verified — a check of the running app could settle it (`sv report --run`); a test of the app's could (ask your AI coding tool to write one)	1	Each function can be used only by people who have been given permission for it.

requirement	status	level	in plain words
V8.2.2	not verified — a check of the running app could settle it (`sv report --run`); a test of the app's could (ask your AI coding tool to write one)	1	Each record can be reached only by people who have been given permission for that record, so nobody can open someone else's by changing an id in the address.
V8.3.1	not verified — only you can (it is in "What only you can check")	1	Who may do what is enforced on the server, never by things a user could change, such as hidden buttons or JavaScript in the browser.
V8.1.2	not verified — only you can (it is in "What only you can check")	2	The authorization documentation sets rules for which fields of a record each user may read and write, by their permissions and the record's attributes.
V8.2.3	not verified — a test of the app's could (ask your AI coding tool to write one)	2	Access to individual fields is restricted to users with explicit permission for those fields (broken object property level authorization, BOPLA).

V11 Cryptography — 14 apply

requirement	status	level	in plain words
V11.3.1	not verified — an outside tool could (they run with `sv report` at a terminal); a test of the app's could (ask your AI coding tool to write one)	1	Insecure encryption modes such as ECB, and weak padding such as PKCS#1 v1.5, are not used.
V11.3.2	checked in part (encryption in ECB mode, or with a retired cipher such as DES, triple DES, RC4, RC2, or Blowfish, in 2 python files (the calls it reads: `ECB`, `MODE_ECB`, `TripleDES`, `ARC4`, `Blowfish`, `CAST5`, `IDEA`, and `SEED`) (ast.weak-cipher))	1	Only approved ciphers and modes, such as AES with GCM, are used.
V11.4.1	checked in part (MD5 or SHA-1 used as a hash function, in 2 python files (the calls it reads: `md5`, `sha1`, and `md4`) (ast.weak-hash-function))	1	Only approved hash functions are used for cryptography; MD5 and the like are never used for any cryptographic purpose.
V11.1.1	not verified — only you can (it is in "What only you can check")	2	A documented policy manages cryptographic keys through their whole lifecycle, following a standard such as NIST SP 800-57, and keys are not shared more widely than needed.
V11.1.2	not verified — only you can (it is in "What only you can check")	2	An up-to-date cryptographic inventory lists every key, algorithm, and certificate the application uses, where each may be used, and what data each may protect.
V11.2.1	not verified — a test of the app's could (ask your AI coding tool to write one)	2	Cryptographic operations use industry-validated implementations, such as well-known libraries or hardware, never home-made ones.
V11.2.2	not verified — a test of the app's could (ask your AI coding tool to write one)	2	The application is built so its cryptographic algorithms, key lengths, and modes can be swapped or upgraded at any time, and keys and passwords replaced and data re-encrypted (crypto agility).
V11.2.3	not verified — an outside tool could (they run with `sv report` at a terminal); a test of the app's could (ask your AI coding tool to write one)	2	Every cryptographic primitive gives at least 128 bits of security, for example a 256-bit ECC key or a 3072-bit RSA key.

requirement	status	level	in plain words
V11.3.3	not verified — a test of the app's could (ask your AI coding tool to write one)	2	Encrypted data is protected against unauthorized changes, preferably with approved authenticated encryption, or with encryption plus an approved MAC.
V11.4.2	**needs attention** (ast.weak-password-key-derivation)	2	Passwords are stored with an approved, deliberately slow key derivation function (a password hashing function), with settings that follow current guidance to make brute-force attacks hard.
V11.4.3	not verified — an outside tool could (they run with `sv report` at a terminal); a test of the app's could (ask your AI coding tool to write one)	2	Hash functions used for digital signatures and data integrity are collision resistant, with outputs of at least 256 bits where collision resistance is needed.
V11.4.4	**needs attention** (ast.weak-password-key-derivation)	2	Keys derived from passwords use an approved key derivation function with key stretching, set strongly enough that brute force cannot recover the key.
V11.5.1	not verified — an outside tool could (they run with `sv report` at a terminal); a test of the app's could (ask your AI coding tool to write one)	2	Every random number or string meant to be unguessable comes from a cryptographically secure random number generator (CSPRNG) and has at least 128 bits of entropy; a UUID does not count.
V11.6.1	not verified — a test of the app's could (ask your AI coding tool to write one)	2	Only approved cryptographic algorithms and modes are used to generate keys and to make and check digital signatures, and no generated key is weak to known attacks.

V12 Secure Communication — 4 apply, 1 do not

requirement	status	level	in plain words
V12.1.1	not verified — an outside tool could (they run with `sv report` at a terminal); a test of the app's could (ask your AI coding tool to write one)	1	Only the latest recommended TLS versions, such as 1.2 and 1.3, are turned on, with the newest preferred.
V12.2.1	not verified — only the live site can show it (`sv probe` once it is on the internet); a test of the app's could (ask your AI coding tool to write one)	1	All connections from clients to the app's public web services use TLS, and never fall back to unencrypted connections.
V12.2.2	not verified — only you can (it is in "What only you can check")	1	The app's public services use TLS certificates that browsers trust.
V12.1.2	not verified — a test of the app's could (ask your AI coding tool to write one)	2	Only recommended TLS cipher suites are enabled, with the strongest preferred.

V13 Configuration — 9 apply

requirement	status	level	in plain words
V13.4.1	not verified — a check of the running app could settle it (`sv report --run`); a test of the app's could (ask your AI coding tool to write one)	1	The app is deployed without its source control folders (.git, .svn), or they cannot be reached, from outside or by the app itself.

requirement	status	level	in plain words
V13.1.1	not verified — only you can (it is in "What only you can check")	2	Every communication need of the application is documented, including the external services it relies on and any place a user could point it to connect to.
V13.2.3	not verified — a test of the app's could (ask your AI coding tool to write one)	2	When a credential is used for one service to authenticate to another, it is never a default credential such as admin/admin.
V13.3.1	not verified — a person has to answer it; supporting: the files this check reads (config.secrets-file-committed); 4 files, against 23 known credential formats plus the assignment rule (secrets.scan)	2	Backend secrets, such as passwords, keys, and API keys, are created, stored, accessed, and destroyed through a secrets management solution such as a key vault, never in the source code or build artifacts.
V13.3.2	not verified — a test of the app's could (ask your AI coding tool to write one)	2	Access to secrets follows the principle of least privilege.
V13.4.2	not verified — a check of the running app could settle it (`sv report --run`); an outside tool could (they run with `sv report` at a terminal); a test of the app's could (ask your AI coding tool to write one)	2	Debug modes are disabled for every component in production, so debugging features and information do not leak.
V13.4.3	not verified — a test of the app's could (ask your AI coding tool to write one)	2	Web servers do not show directory listings unless that is explicitly intended.
V13.4.4	not verified — a check of the running app could settle it (`sv report --run`); a test of the app's could (ask your AI coding tool to write one)	2	The HTTP TRACE method is not supported in production, so it cannot leak information.
V13.4.5	not verified — only you can (it is in "What only you can check")	2	Documentation, such as for internal APIs, and monitoring endpoints are not exposed unless that is explicitly intended.

V14 Data Protection — 9 apply

requirement	status	level	in plain words
V14.2.1	not verified — a test of the app's could (ask your AI coding tool to write one)	1	Sensitive data, such as an API key or a session token, is sent only in the body or the headers of a request, never in the web address.
V14.3.1	not verified — a check of the running app could settle it (`sv report --run`); a test of the app's could (ask your AI coding tool to write one)	1	Signed-in data is cleared from the browser when the session ends, even when the server cannot be reached.
V14.1.1	not verified — only you can (it is in "What only you can check")	2	All sensitive data the application creates or handles is identified and classified into protection levels, including data only encoded (such as Base64 or a JWT's payload), with the regulations it must follow in mind.
V14.1.2	not verified — only you can (it is in "What only you can check")	2	Each protection level for sensitive data has documented requirements: encryption, integrity, retention, logging, access to logs, database encryption, privacy, and other confidentiality needs.
V14.2.2	not verified — only you can (it is in "What only you can check")	2	Sensitive data is not cached in server components such as load balancers and application caches, or it is securely purged after use.

requirement	status	level	in plain words
V14.2.3	not verified — only you can (it is in "What only you can check")	2	Sensitive data is never sent to untrusted parties, such as user trackers, where it would be collected outside the application's control.
V14.2.4	not verified — only you can (it is in "What only you can check")	2	The controls for sensitive data (encryption, integrity, retention, logging, privacy) are implemented as the documentation sets them for that data's protection level.
V14.3.2	not verified — a check of the running app could settle it (`sv report --run`); a test of the app's could (ask your AI coding tool to write one)	2	Responses carrying sensitive data set anti-caching headers (Cache-Control: no-store), so browsers do not cache it.
V14.3.3	not verified — a test of the app's could (ask your AI coding tool to write one)	2	Browser storage (localStorage, sessionStorage, IndexedDB, or cookies) holds no sensitive data, apart from session tokens.

V15 Secure Coding and Architecture — 12 apply

requirement	status	level	in plain words
V15.1.1	not verified — only you can (it is in "What only you can check")	1	The documentation sets deadlines for fixing third-party components with known weaknesses, and for keeping libraries up to date, by how risky they are.
V15.2.1	not verified — the advisory data could, once it is downloaded	1	Every component the app ships is within the update and fix deadlines its documentation sets.
V15.3.1	not verified — a test of the app's could (ask your AI coding tool to write one)	1	The app returns only the fields of a record that are needed, never the whole record with fields some users must not see.
V15.1.2	not verified — sv's own reading of the code looked, and settled nothing; a test of the app's could (ask your AI coding tool to write one)	2	An inventory, such as a software bill of materials (SBOM), lists every third-party library in use, and checks each comes from a trusted, maintained repository.
V15.1.3	not verified — only you can (it is in "What only you can check")	2	The documentation names the functions that are slow or use many resources, and how to keep them from making the app unavailable or taking longer than the consumer's timeout.
V15.2.2	not verified — only you can (it is in "What only you can check")	2	The application defends against slow or resource-heavy functions making it unavailable, as the documented decisions set out.
V15.2.3	not verified — a test of the app's could (ask your AI coding tool to write one)	2	The production environment holds only the functionality the application needs, with no test code, sample snippets, or development features exposed.
V15.3.3	not verified — an outside tool could (they run with `sv report` at a terminal); a test of the app's could (ask your AI coding tool to write one)	2	The application protects against mass assignment by limiting which fields each action may set, so a field not meant to be changed by an action cannot be.
V15.3.4	not verified — a test of the app's could (ask your AI coding tool to write one)	2	Proxies and middleware pass on the user's original IP address in trusted fields the user cannot manipulate, and the app uses that value for logging and rate limiting.
V15.3.5	not verified — a test of the app's could (ask your AI coding tool to write one)	2	The code checks variables are of the correct type and uses strict equality and comparisons, so type juggling or type confusion cannot be exploited.

requirement	status	level	in plain words
V15.3.6	not verified — a test of the app's could (ask your AI coding tool to write one)	2	JavaScript code is written to prevent prototype pollution, for example by using Map() or Set() instead of plain object literals.
V15.3.7	not verified — a test of the app's could (ask your AI coding tool to write one)	2	The application defends against HTTP parameter pollution, especially where its framework does not tell apart parameters from the query string, body, cookies, and header fields.

V16 Security Logging and Error Handling — 15 apply

requirement	status	level	in plain words
V16.1.1	not verified — only you can (it is in "What only you can check")	2	An inventory documents the logging at each layer: what events are logged, the formats, where logs are stored, how they are used, who can access them, and how long they are kept.
V16.2.1	not verified — a check of the running app could settle it (`sv report --run`); a test of the app's could (ask your AI coding tool to write one)	2	Each log entry has the metadata needed to investigate an event's timeline: when, where, who, and what.
V16.2.2	not verified — a check of the running app could settle it (`sv report --run`); a test of the app's could (ask your AI coding tool to write one)	2	All logging components use synchronized time sources, and security event timestamps are in UTC or carry an explicit time zone offset.
V16.2.3	not verified — only you can (it is in "What only you can check")	2	Logs are stored or sent only to the files and services the log inventory documents.
V16.2.4	not verified — a check of the running app could settle it (`sv report --run`); a test of the app's could (ask your AI coding tool to write one)	2	Logs can be read and correlated by the log processor in use, preferably by using a common logging format.
V16.2.5	not verified — an outside tool could (they run with `sv report` at a terminal); a test of the app's could (ask your AI coding tool to write one)	2	Sensitive data is logged only as its protection level allows: some, such as credentials or payment details, never; others, such as session tokens, only hashed or masked.
V16.3.1	not verified — a check of the running app could settle it (`sv report --run`); a test of the app's could (ask your AI coding tool to write one)	2	Every authentication operation is logged, successful and unsuccessful, preferably with the type of authentication or factors used.
V16.3.2	not verified — a check of the running app could settle it (`sv report --run`); a test of the app's could (ask your AI coding tool to write one)	2	Failed authorization attempts are logged.
V16.3.3	not verified — a check of the running app could settle it (`sv report --run`)	2	The security events the documentation defines are logged, as are attempts to bypass security controls such as input validation, business logic, and anti-automation.
V16.3.4	not verified — a check of the running app could settle it (`sv report --run`); a test of the app's could (ask your AI coding tool to write one)	2	Unexpected errors and security control failures, such as a backend TLS failure, are logged.
V16.4.1	not verified — a test of the app's could (ask your AI coding tool to write one)	2	Every logging component encodes data appropriately to prevent log injection.

requirement	status	level	in plain words
V16.4.2	not verified — a test of the app's could (ask your AI coding tool to write one)	2	Logs are protected from unauthorized access and cannot be modified.
V16.4.3	not verified — only you can (it is in "What only you can check")	2	Logs are sent securely to a logically separate system for analysis and alerting, so a breach of the application does not compromise them.
V16.5.1	not verified — a check of the running app could settle it (`sv report --run`); a test of the app's could (ask your AI coding tool to write one)	2	When an unexpected or security-sensitive error occurs, the app returns a generic message, never internal details such as stack traces, queries, secret keys, or tokens.
V16.5.3	not verified — a test of the app's could (ask your AI coding tool to write one)	2	The application fails gracefully and securely, including on an exception, so it never fails open, such as completing a transaction despite a validation error.

SBD-AC Access Control & Secure Communication — 5 apply

requirement	status	level	in plain words
SBD-AC-01	not verified — only you can: no check of sv's speaks to it	1	All communications use TLS or mutual TLS (mTLS), and each service's identity is verified, for example with certificates issued by a service mesh.
SBD-AC-02	not verified — only you can: no check of sv's speaks to it	1	A central identity provider (IdP, using OIDC or OAuth2) is used, multi-factor authentication (MFA) is enforced on privileged and admin paths, and tokens are short-lived.
SBD-AC-03	not verified — only you can: no check of sv's speaks to it	1	Role- or attribute-based access control (RBAC or ABAC) governs the APIs and who may publish or consume messages, and least privilege (only the access needed) is applied.
SBD-AC-05	not verified — a person has to answer it; supporting: 4 files, against 23 known credential formats plus the assignment rule (secrets.scan); the files this check reads, as evidence about V13.3.1 (config.secrets-file-committed)	1	Secrets are stored in a secret manager, keys and certificates rotate automatically, and no secrets appear in code or logs.
SBD-AC-06	not verified — only you can: no check of sv's speaks to it	1	The regulatory controls that apply, such as privacy or financial rules, are identified, with evidence that the design aligns with them.

SBD-AS Architecture & Service Design — 1 apply

requirement	status	level	in plain words
SBD-AS-07	not verified — only you can: no check of sv's speaks to it	2	Services start up safely when a dependency is missing, and autoscaling (adding or removing copies as load changes) is defined with a minimum number of replicas.

SBD-DM Data Management & Protection — 3 apply

requirement	status	level	in plain words
SBD-DM-02	not verified — only you can: no check of sv's speaks to it	1	Encryption is designed in, both in transit (TLS or mutual TLS) and at rest, with managed keys and key rotation.
SBD-DM-01	not verified — only you can: no check of sv's speaks to it	2	Data are classified, each class has named owners, and the controls protecting data are proportional to its classification.
SBD-DM-05	not verified — only you can: no check of sv's speaks to it	2	Each class of data has retention and deletion policies, and minimization is applied, so only the data needed is collected and stored.

SBD-MT Monitoring, Testing & Incident Readiness — 7 apply

requirement	status	level	in plain words
SBD-MT-02	not verified — only you can: no check of sv's speaks to it	1	Metrics, service level objectives (SLOs), and dashboards cover service health and anomalies in event flow, and alerts are actionable.
SBD-MT-03	not verified — only you can: no check of sv's speaks to it	1	The test strategy plans security tests aligned with ASVS, and negative tests (checking that what should fail does fail) drawn from threat modeling.
SBD-MT-04	not verified — only you can: no check of sv's speaks to it	1	Secure by Design (SbD) controls are verifiable in tests, including isolation, mTLS, authorization denials, and rate-limit behavior.
SBD-MT-05	not verified — only you can: no check of sv's speaks to it	1	API descriptions (OpenAPI and AsyncAPI) and event catalogs are published, and decision records (ADRs) and runbooks are kept current.
SBD-MT-06	not verified — only you can: no check of sv's speaks to it	1	A documented Incident Response plan, covering roles, communications, and evidence handling, exists and is rehearsed.
SBD-MT-07	not verified — only you can: no check of sv's speaks to it	1	Audit log retention of at least 12 months is defined, with tamper-evidence (changes can be detected) and a searchable hot window of recent logs.
SBD-MT-01	not verified — only you can: no check of sv's speaks to it	2	Logs are structured and centralized, carry correlation or trace IDs that link related entries, and administrative access is logged.

SBD-RR Reliability & Resilience — 5 apply

requirement	status	level	in plain words
SBD-RR-06	not verified — only you can: no check of sv's speaks to it	1	All calls have timeouts, services have health probes, and critical paths have explicit failover strategies, such as running across multiple availability zones (multi-AZ).
SBD-RR-08	not verified — only you can: no check of sv's speaks to it	1	Caching, a CDN (content delivery network), and back-pressure (slowing senders when the system is overloaded) are applied where appropriate, and performance guardrails are defined.
SBD-RR-01	not verified — only you can: no check of sv's speaks to it	2	Retries wait longer after each failure (exponential backoff) plus a small random delay (jitter), and default exception handling and safe error models exist.
SBD-RR-05	not verified — only you can: no check of sv's speaks to it	2	Every endpoint that changes data enforces an Idempotency-Key (so a repeated request is not applied twice), and concurrency controls, such as locks or optimistic concurrency, are in place.
SBD-RR-07	not verified — only you can: no check of sv's speaks to it	2	Quotas and autoscaling are defined, rate limits are enforced at the edges where traffic comes in, and the minimum capacity is documented.

C7 Model Behavior, Output Control & Safety Assurance — 2 apply

requirement	status	level	in plain words
C7.2.1	not verified — only you can (it is in "What only you can check")	2	The system assesses how reliable generated answers are using a confidence estimation method.
C7.2.2	not verified — only you can (it is in "What only you can check")	2	The application automatically blocks an answer, or switches to a fallback message, when its confidence score drops below a defined threshold.

C12 Monitoring, Logging & Anomaly Detection — 3 apply

requirement	status	level	in plain words
C12.2.2	not verified — only you can (it is in "What only you can check")	2	Behavioral anomaly detection identifies unusual conversation patterns, excessive retry attempts, or probing behaviors.
C12.2.3	not verified — only you can (it is in "What only you can check")	2	Custom rules detect AI-specific threat patterns for coordinated jailbreak attempts, prompt injection, and attempts to extract the system prompt (the AI's hidden instructions).
C12.2.4	not verified — a test of the app's could (ask your AI coding tool to write one)	2	Extraction-alert events include metadata about the offending queries to support investigation.

How the app is built with AI (OWASP AISVS Appendix C)

33 requirements of OWASP AISVS Appendix C (AI-Assisted Secure Coding) apply to how this app is built with an AI coding tool, rather than to the app itself, and nothing has checked them. They are listed here rather than in the counts above. 23 are what the rules given to your AI coding tool come from (`sv rules`, or `stackvet_guidance` from inside the tool); the rules are instructions, and following them is not evidence that they are met. 2 are your decisions, and are among your questions. Nothing in `sv` reaches the other 8: most are about a CI pipeline or an organization's AI tooling. 19 more wait on a question nobody has answered.

requirement	what happens to it	what it asks for
AC.1.1	given to your AI coding tool as a rule	Verify that a written workflow says when AI tools may generate, refactor, or review code. The workflow names the approved tools, the prohibited use cases, and the data classifications that are allowed as input.
AC.1.2	given to your AI coding tool as a rule	Verify that the workflow covers every SSDLC phase from design and implementation through code review, testing, deployment, and post-deployment monitoring, and names the security gates that stay mandatory whether AI was involved or not.
AC.1.3	nothing in `sv` reaches it	Verify that the workflow names the adversarial-AI threat scenarios it is built to mitigate. The list should cover prompt injection delivered through PR content, AI-generated supply-chain payloads, autonomous agents approving their own work, fork-PR secret exfiltration, and compromise of the model supply chain.
AC.2.1	given to your AI coding tool as a rule	Verify that every AI tool, whether it is an assistant, a reviewer, an agent, or an MCP server, has a threat model. The threat model covers misuse, model inversion, training-data leakage, prompt injection from untrusted input, insecure output handling, excessive agency, and risk inherited from its dependency chain.

requirement	what happens to it	what it asks for
AC.2.2	given to your AI coding tool as a rule	Verify that the evaluation of each tool covers the local components (static and dynamic analysis), the SaaS endpoints (TLS, AuthN/AuthZ, logging, data residency), and the vendor's model supply chain (training-data provenance, fine-tune history, RAG sources). Each of these is reviewed and the review is written down.
AC.2.3	nothing in sv reaches it	Verify that each tool goes through adversarial robustness testing before onboarding. The testing is repeated after any material change to the model or to the system prompts. Coverage includes automated prompt-injection probes, jailbreak suites, and indirect-injection corpora delivered through realistic PR and issue surfaces.
AC.3.1	given to your AI coding tool as a rule	Verify that written guidance forbids putting secrets, credentials, PII, or classified data in any prompt sent to an AI tool. The guidance is enforced in pre-commit hooks, IDE integrations, and CI.
AC.3.2	given to your AI coding tool as a rule	Verify that technical controls automatically strip sensitive material from any context window sent to an AI tool. Client-side redaction, approved context filters, and secret scanners with pre-prompt hooks all qualify.
AC.3.3	given to your AI coding tool as a rule	Verify that any externally sourced context being fed to an AI tool is treated as untrusted and screened for prompt injection before it reaches the prompt. Sources to cover: PR descriptions and comments, fork-supplied diffs, issue bodies, commit messages, third-party documentation, web search results, and MCP tool outputs.
AC.3.4	given to your AI coding tool as a rule	Verify that the AI tool enforces an instruction hierarchy, with system and developer messages taking precedence over untrusted repository content. This hierarchy has to hold across multi-turn conversations and tool-augmented workflows.
AC.3.5	given to your AI coding tool as a rule	Verify that input length controls stop untrusted PR or repository content from crowding system instructions or safety directives out of the effective context window. Oversized inputs are rejected outright. Silent truncation is not acceptable.
AC.4.1	your decision, among your questions	Verify that AI-generated code always goes through code review by a qualified human engineer. The reviewer must not be the same identity that asked for the AI generation in the first place (separation of duties). And the AI agent itself does not count as the human reviewer.
AC.4.2	given to your AI coding tool as a rule	Verify that automated security testing runs on every pull request containing AI-generated code: SAST, IAST, DAST, secret scanning, IaC scanning, and SCA. Where the scanner supports them, AI-attribution-aware rules are turned on.
AC.4.3	given to your AI coding tool as a rule	Verify that pull requests containing AI-generated code are blocked from merging when an automated scan surfaces a critical security finding, defined as CVSS >= 9.0 or the equivalent threshold in the organization's vulnerability severity policy. Bypassing the block requires a written exception approved by an authorized human.
AC.4.4	given to your AI coding tool as a rule	Verify that security-critical files require an elevated review threshold when AI generated or modified them: two-person review, security-team sign-off, or stricter. Security-critical files here include authentication, authorization, and cryptography code; IAM policy; CI/CD workflow definitions; deployment manifests; and sandbox or network policy artifacts.
AC.5.1	nothing in sv reaches it	Verify that prompt-and-response pairs are logged with stable correlation identifiers, so that an investigator can later replay the whole chain: prompt -> response -> commit -> build -> deployment.
AC.6.3	your decision, among your questions	Verify that scheduled red-team exercises target the AI tooling itself. The exercises include direct and indirect prompt-injection probes delivered through realistic PR, issue, and comment surfaces, jailbreak corpora, and supply-chain payload generation. Findings are remediated under tracked severity SLAs.

requirement	what happens to it	what it asks for
AC.7.1	given to your AI coding tool as a rule	Verify that AI-generated or AI-modified artifacts are clearly labeled and tracked as such. Artifact classes in scope include infrastructure-as-code (Terraform, CloudFormation, Pulumi, Bicep), CI/CD workflow files (GitHub Actions, GitLab CI, Jenkinsfile, Argo Workflows, Tekton), container and orchestration manifests (Dockerfile, Kubernetes, Helm), and security policy artifacts (IAM, OPA/Rego, NetworkPolicy, admission controllers).
AC.7.2	given to your AI coding tool as a rule	Verify that AI-generated infrastructure and pipeline configurations require human review and approval before they run in any environment beyond a hermetic sandbox.
AC.8.1	given to your AI coding tool as a rule	Verify that autonomous agents cannot approve, merge, sign, or deploy artifacts that they themselves generated, and that this constraint is enforced by the source-control system, the CI system, and the artifact registry. Policy alone does not satisfy this control.
AC.8.2	given to your AI coding tool as a rule	Verify that AI systems run with scoped, non-human identities (service accounts, workload identities, OIDC-issued ephemeral tokens), and that those identities cannot be used to promote their own generated artifacts across environments.
AC.10.1	given to your AI coding tool as a rule	Verify that AI-generated artifacts carry the required origin and generation fields: model identity and version, tool or agent identity, generation context, prompt hash, human involvement, session identifiers, and correlation IDs.
AC.10.2	nothing in sv reaches it	Verify that origin and generation metadata is checked for completeness and consistency: no missing or ambiguous fields, values normalized to a single representation, and a signature chain that validates back to a trusted root.
AC.11.1	given to your AI coding tool as a rule	Verify that AI review and assistant bots treat every piece of PR-supplied content (diff, title, description, comments, file contents, commit messages, linked external URLs) as untrusted input, and apply the AISVS C2.1 prompt-injection defenses: instruction-hierarchy enforcement, content sanitization, and indirect-injection detection.
AC.11.2	given to your AI coding tool as a rule	Verify that AI review and assistant bot system prompts and policy configurations are integrity-checked at load time (signed, hash-pinned), and that nothing in the repository, in branch contents, in PR-sourced environment variables, or in any other user-controllable input can modify them.
AC.11.3	given to your AI coding tool as a rule	Verify that AI review and assistant bots emit only structured, schema-validated output (JSON with an allow-list of fields and actions). Any free-form output is treated as untrusted and never executed as a command, a query, a shell snippet, or a workflow step.
AC.11.4	nothing in sv reaches it	Verify that AI review and assistant bots run in network-isolated, least-privilege sandboxes: a dedicated namespace, default-deny egress with an allow-list to approved APIs only, no mounted repository secrets, and ephemeral credentials only.
AC.11.5	given to your AI coding tool as a rule	Verify that any privileged action a bot can take (approving a PR, merging, labeling, dismissing reviews, posting comments outside its sandbox, invoking external tools) goes through a separate, audited authorization path. That path is adjudicated by a policy engine, not by the LLM.
AC.11.6	nothing in sv reaches it	Verify that AI review and assistant bots log all prompts (including externally sourced context), tool calls, and outputs to tamper-evident storage. Egress patterns (URLs, IPs, DNS, payload sizes) are continuously monitored for exfiltration indicators, with alerting tuned for webhook, paste-site, and bin-service destinations.
AC.14.1	given to your AI coding tool as a rule	Verify that an incident-response playbook exists for AI-in-pipeline compromise. At minimum it covers: revoking AI-agent credentials, rotating every secret that touched the compromised workflow run, quarantining the compromised artifacts, notifying downstream consumers, notifying regulators where applicable, and preserving prompts, responses, and audit logs for forensics.

requirement	what happens to it	what it asks for
AC.14.2	given to your AI coding tool as a rule	Verify that any secret that touched a workflow run associated with a suspicious PR, a prompt-injection event, or an AI-agent anomaly is automatically rotated, and that downstream issuers (cloud IAM, package registries, signing-key custodians) are notified of the rotation.
AC.14.3	nothing in `sv` reaches it	Verify that AI agent identities (keys, tokens, OIDC trust grants) can be rapidly revoked and quarantined, with a target time-to-revoke that is written down and tested at least once a year.
AC.14.4	nothing in `sv` reaches it	Verify that build provenance and AI BOM records are used during incident response to identify every downstream artifact produced under the suspect AI agent or the compromised pipeline run, so that recall, rebuild, or quarantine can be targeted.

Threats

What could go wrong with an app like this one, by the part of it each threat concerns, and what the checks showed about each. These are the threats sv has rules for, for what was found in this app: not every threat there is. None is called handled. A threat is only as settled as the requirements that answer it, and each of those is one automated check at most, so the best a threat can be here is checked in part. Found threats come first, then those nothing has looked at.

Parts of the app: Browsers and the app; The app itself; The app's stored data; The AI model (not known: stackvet.toml does not answer `ai`); Uploaded files; Email (not known: stackvet.toml does not answer `email`); Payments (not known: stackvet.toml does not answer `payments`); Programs using the app's API (not known: stackvet.toml does not answer `public-api`); Outside services (not known: stackvet.toml does not answer `external-apis`); Scheduled jobs (not known: stackvet.toml does not answer `scheduler`); Tools the model can use (MCP) (not known: stackvet.toml does not answer `ai`, `mcp`); The documents the model searches (not known: stackvet.toml does not answer `ai`, `rag`); The app's own services (not known: stackvet.toml does not answer `multiple-services`).

40 threats: 1 found, 11 not verified, 2 checked in part, 26 cannot place.

threat	status	part of the app	what could happen	the requirements that answer it
T-25 (exposing information)	found	The app's stored data	Someone with access to the computer, or to a backup, copies the stored data and reads it.	needs attention: V11.4.2; not verified: V14.2.4
T-01 (pretending to be someone else)	not verified	Browsers and the app	Someone signs in as another person by guessing a password, reusing a leaked one, or stealing a session cookie.	not verified: V6.2.1, V6.2.4, V6.3.1, V7.2.3, V3.3.4
T-02 (gaining more access than allowed)	not verified	Browsers and the app	A signed-in person opens administrator pages or other people's records by changing a web address.	not verified: V8.2.1, V8.3.1, V8.2.2, V8.2.3, V15.3.1
T-03 (denying having done something)	not verified	Browsers and the app	Someone denies having signed in or changed something, and there is no reliable record of it.	not verified: V16.3.1, V16.3.2, V16.4.2
T-05 (exposing information)	not verified	Browsers and the app	Traffic is read on the network, or pages, caches, and error messages reveal what they should not.	not verified: V16.5.1, V14.3.2, V14.3.1, V12.1.1, V3.4.1, V3.3.1

threat	status	part of the app	what could happen	the requirements that answer it
T-06 (making the app unavailable)	not verified	Browsers and the app	Floods of requests, slow connections, or huge request bodies make the app unavailable.	not verified: V2.4.1, V15.2.2
T-27 (changing what should not change)	not verified	The app itself	A package the app depends on has a known vulnerability or a malicious update.	not verified: V15.1.2, V15.2.1
T-28 (exposing information)	not verified	The app itself	Secrets (session keys, encryption keys, API keys) leak through the code, the logs, or error pages.	not verified: V13.3.1, V16.2.5, V13.4.1, V16.5.1
T-29 (denying having done something)	not verified	The app itself	Log entries are altered or deleted to hide what happened.	not verified: V16.4.2, V16.4.1
T-26 (changing what should not change)	not verified	The app's stored data	A form submitted twice, or two people editing at once, corrupts or duplicates a record.	not verified: V2.3.3
T-14 (making the app unavailable)	not verified	Uploaded files	Oversized or endless uploads fill the disk.	not verified: V5.2.1, V2.3.2
T-15 (exposing information)	not verified	Uploaded files	Someone downloads another person's file by guessing its address.	not verified: V8.2.2
T-04 (changing what should not change)	checked in part: 1 of 8 requirements checked	Browsers and the app	Crafted input (injection, a request forged from another site, oversized or unexpected fields) changes data or behavior.	checked: V1.2.4; not verified: V2.2.1, V2.2.2, V15.3.3, V1.2.1, V3.4.3, V3.5.1, V15.3.6
T-13 (changing what should not change)	checked in part: 0 of 5 requirements checked, 1 checked in part	Uploaded files	A harmful file, such as a script disguised as an image or an SVG with code in it, is uploaded and later opened by someone else.	checked: V5.3.2; not verified: V5.2.2, V5.3.1, V5.4.1, V5.4.2
T-41 (exposing information)	cannot place	The app's stored data	One customer organization (tenant) sees or changes another tenant's records in a system they share.	whether this applies is not known: answer multi-tenant in stackvet.toml
T-07 (changing what should not change)	cannot place	The AI model	Text typed by a person, or hidden inside a record, tricks the model into ignoring its instructions (prompt injection).	whether this applies is not known: answer ai in stackvet.toml
T-08 (exposing information)	cannot place	The AI model	The model reveals other people's records, or its own instructions.	whether this applies is not known: answer ai in stackvet.toml
T-09 (making the app unavailable)	cannot place	The AI model	Runaway or abusive use of the model runs up the AI bill, spends tokens without limit, or exhausts the provider's quota.	whether this applies is not known: answer ai in stackvet.toml
T-10 (pretending to be someone else)	cannot place	The AI model	The AI provider's API key leaks, in logs, in a prompt, or in generated text, and someone else uses it.	whether this applies is not known: answer ai in stackvet.toml
T-11 (gaining more access than allowed)	cannot place	The AI model	The model makes a change the person did not intend, or one they are not allowed to make.	whether this applies is not known: answer ai, ai-actions in stackvet.toml

threat	status	part of the app	what could happen	the requirements that answer it
T-12 (changing what should not change)	cannot place	The AI model	The model is used to produce or spread harmful content.	whether this applies is not known: answer ai, ai-moderation in stackvet.toml
T-16 (changing what should not change)	cannot place	Email	Someone adds hidden recipients or content to an email through a form field.	whether this applies is not known: answer email in stackvet.toml
T-17 (pretending to be someone else)	cannot place	Email	Password reset emails are abused to take over an account.	whether this applies is not known: answer email in stackvet.toml
T-18 (exposing information)	cannot place	Payments	Card numbers end up stored or logged by the app.	whether this applies is not known: answer payments in stackvet.toml
T-19 (changing what should not change)	cannot place	Payments	A fake "payment succeeded" message marks an order as paid.	whether this applies is not known: answer payments in stackvet.toml
T-20 (pretending to be someone else)	cannot place	Programs using the app's API	A leaked API key is used by someone else.	whether this applies is not known: answer public-api in stackvet.toml
T-21 (making the app unavailable)	cannot place	Programs using the app's API	A program hammers the API with requests.	whether this applies is not known: answer public-api in stackvet.toml
T-22 (exposing information)	cannot place	Outside services	Data sent to an outside service is exposed if that service is breached or the connection is intercepted.	whether this applies is not known: answer external-apis in stackvet.toml
T-23 (making the app unavailable)	cannot place	Outside services	An outside service is slow or down and takes the app down with it.	whether this applies is not known: answer external-apis in stackvet.toml
T-24 (changing what should not change)	cannot place	Outside services	The app is tricked into calling a different address than the service it means to (server-side request forgery).	whether this applies is not known: answer external-apis in stackvet.toml
T-30 (changing what should not change)	cannot place	Scheduled jobs	A scheduled job runs twice at once, or a long one takes the app's resources and makes it unavailable.	whether this applies is not known: answer scheduler in stackvet.toml
T-31 (changing what should not change)	cannot place	Tools the model can use (MCP)	A tool's description or response carries hidden instructions that the model follows (indirect prompt injection).	whether this applies is not known: answer ai, mcp in stackvet.toml
T-32 (gaining more access than allowed)	cannot place	Tools the model can use (MCP)	The model calls a tool, or passes it arguments, that the signed-in person is not allowed to use.	whether this applies is not known: answer ai, mcp in stackvet.toml
T-33 (pretending to be someone else)	cannot place	Tools the model can use (MCP)	Someone reaches the app's MCP server without a valid token, or with a token meant for another service.	whether this applies is not known: answer ai, mcp in stackvet.toml
T-34 (changing what should not change)	cannot place	Tools the model can use (MCP)	A malicious or look-alike MCP server is installed, or a local one reaches files and networks it should not.	whether this applies is not known: answer ai, mcp in stackvet.toml

threat	status	part of the app	what could happen	the requirements that answer it
T-35 (exposing information)	cannot place	The documents the model searches	The model answers one person with documents another person may not see, because retrieval does not check who is asking.	whether this applies is not known: answer ai, rag in stackvet.toml
T-36 (changing what should not change)	cannot place	The documents the model searches	Documents written into the store, including the model's own output, steer its later answers.	whether this applies is not known: answer ai, rag in stackvet.toml
T-37 (exposing information)	cannot place	The documents the model searches	Sensitive fields are embedded into the store, where any search can bring them back.	whether this applies is not known: answer ai, rag in stackvet.toml
T-42 (exposing information)	cannot place	The documents the model searches	One tenant's documents come back in another tenant's searches.	whether this applies is not known: answer ai, rag, multi-tenant in stackvet.toml
T-38 (pretending to be someone else)	cannot place	The app's own services	One of the app's own services is called by something that is not another of its services, because the calls between them are not authenticated.	whether this applies is not known: answer multiple-services in stackvet.toml

For a security reviewer: these threats in MITRE ATLAS

The threats above that are about AI, as MITRE ATLAS names them (release 2026.09), for anyone who works from that catalog. These are references, not checks: nothing was tested against ATLAS, and they change no status above. MITRE ATLAS is a trademark of The MITRE Corporation; its data is used under the Apache License 2.0.

threat	MITRE ATLAS technique	what the two share
T-07	AML.T0051 LLM Prompt Injection	prompt injection
T-08	AML.T0057 LLM Data Leakage	data leakage of other people's records
T-08	AML.T0056 Extract LLM System Prompt	revealing its own instructions, the system prompt
T-09	AML.T0034 Cost Harvesting	running up the bill: the cost, harvesting it
T-09	AML.T0029 Denial of AI Service	exhausting the provider's quota, a denial of the service
T-10	AML.T0055 Unsecured Credentials	a leaked API key is an unsecured credential
T-11	AML.T0053 AI Agent Tool Invocation	the model makes a change through an agent tool invocation
T-12	AML.T0048 External Harms	harmful content spread to others, external harms

What only you can check

49 of the requirements that apply cannot be settled by any tool: they ask what your rules are, how the app is built, or what is true of it in production. Each one below says what doing something about it involves. None of them is counted as met — doing the thing is what would change that, not reading it here.

A further 53 are the Secure by Design and AISVS design-review controls, which are not listed one by one: those standards are checklists already, and repeating them here would be another wall of text. They are in the table above, and in the standards themselves.

requirement	what to do	how
V2.1.1 — What counts as valid input	tell your AI coding tool your answer, and it writes it in security-notes.md (at a terminal, `sv notes` makes the file)	For each thing people type into the app (a name, an email address, a date, an amount), what does a good one look like, how long may it be, and what should the app do with one that is wrong? In the standard's words: For each kind of information people type into the app (names, email addresses, dates, amounts, free text), write down the validation rules: what a valid value looks like, how long it may be, and what the app does with one that is not valid. **Where to look:** Go through every form and every field an API accepts, and for each one write what a good value looks like. The quickest way to find them all is to open each page that has a form and list its inputs.
V2.1.2 — Data items that must make sense combined	tell your AI coding tool your answer, and it writes it in security-notes.md (at a terminal, `sv notes` makes the file)	Which things people enter have to make sense together (a start date before an end date, say), and how does the app check them? In the standard's words: Where two or more data items have to agree with each other once combined (a start date before an end date, a city that matches the ZIP code), write down which combinations matter and how the app checks that logical consistency. **Where to look:** Open each form that takes more than one value and list the pairs that have to agree — two dates, a country and a postal code, a quantity and a maximum. For each, try the combination that should be impossible and see whether the app takes it.
V2.1.3 — Business limits	tell your AI coding tool your answer, and it writes it in security-notes.md (at a terminal, `sv notes` makes the file)	How many of something may one person make or do, and in how long? And is there any limit for the whole app, such as a total per day? In the standard's words: Write down the business logic limits: how many of something one person may create or do, and in how long (per user), and any limits across the whole app (globally), such as a daily total. **Where to look:** Go through the app as a user and note everything somebody could do repeatedly: create a record, send an invitation, start an export. For each, decide the number that would be too many in a day, and write that down.
V5.1.1 — What files people may upload	tell your AI coding tool your answer, and it writes it in security-notes.md (at a terminal, `sv notes` makes the file)	Where people can upload a file: which kinds of file are allowed, how big may one be (unpacked, for a zip), and what does the app do with a file that looks dangerous? In the standard's words: For each place people can upload a file: the permitted file types and extensions, the maximum size (including once unpacked, for a zip), and how the app makes a file safe to download again, including what happens when a file looks malicious. **Where to look:** Open each place the app takes a file and look at what the code accepts today: the extensions it allows, the size limit if there is one, and where the file ends up. Write down what it should be, which may not be what it is.
V6.1.1 — How sign-in is protected against guessing	tell your AI coding tool your answer, and it writes it in security-notes.md (at a terminal, `sv notes` makes the file)	After how many wrong passwords in a row does the app slow down or lock the account, and for how long? And how does it stop somebody locking a real person out on purpose? (Say the numbers you want, even if the app does not do it yet.) In the standard's words: How the app defends against someone trying many passwords (password brute force) or leaked email and password pairs (credential stuffing): rate limiting, delays, or lockout, and the numbers, such as five failed attempts in fifteen minutes. Say how these controls are configured, and how the app stops an attacker locking a real person out on purpose (malicious account lockout). **Where to look:** Look in your sign-in code for anything that counts failures — a rate limiter, a lockout, a delay. If you find nothing, that is the answer, and the number to write down is the one you want rather than the one you have.
V6.1.2 — Words not allowed in passwords	tell your AI coding tool your answer, and it writes it in security-notes.md (at a terminal, `sv notes` makes the file)	Which words should nobody be allowed to use in a password here, because they are the first anyone would guess (the app's name, your organization's name, your products)? In the standard's words: A list of context-specific words people may not use in their passwords: the app's name, the organization's name, product names, and other words someone would guess first. **Where to look:** Start from the names somebody could guess without knowing you: the app, your organization, your product, the town you are in. Add the obvious variations with numbers on the end.

requirement	what to do	how
V6.1.3 — Every way to sign in	tell your AI coding tool your answer, and it writes it in security-notes.md (at a terminal, `sv notes` makes the file)	What are all the ways someone can sign in (a password, a Google or Microsoft account, a code by email, a key for programs), and are they all protected as well as each other? In the standard's words: If there is more than one way to sign in (a password, a Google or Microsoft account, a code by email, an API key), list all these authentication pathways, and say which security controls apply to each, so they are enforced consistently and one is not weaker than the others. **Where to look:** Open the sign-in page and list every button on it, then search the code for anywhere else a session is created — an API key check, a magic link, a mobile app, an admin back door. Each one is a pathway, and they should all be as strong as each other.
V7.1.1 — How long someone stays signed in	tell your AI coding tool your answer, and it writes it in security-notes.md (at a terminal, `sv notes` makes the file)	How long may someone be away before they have to sign in again, how long may anyone stay signed in at most, and why are those times right for this app? In the standard's words: The session inactivity timeout (how long someone can be away before signing in again) and the absolute maximum session lifetime (how long any session lasts at most), and why those times are right for this app. If they are longer than NIST SP 800-63B suggests, say why. **Where to look:** Find the two numbers in your app's configuration: how long an idle session lasts, and how long any session lasts at most. If you cannot find them, the answer is whatever your framework does by default, and that is the number to write down and then reconsider.
V7.1.2 — Signed in on several devices at once	tell your AI coding tool your answer, and it writes it in security-notes.md (at a terminal, `sv notes` makes the file)	How many devices may one person be signed in on at once, and what happens when they sign in on one more? In the standard's words: How many concurrent sessions one account may have at once (a phone and a laptop, say), and what the app does when someone reaches the maximum: sign out the oldest, refuse the new one, or tell the person. **Where to look:** Sign in to the same account from two browsers and see whether both keep working. That tells you what the app does today; write down what it should do.
V8.1.1 — Who may do what	tell your AI coding tool your answer, and it writes it in security-notes.md (at a terminal, `sv notes` makes the file)	Who may do what in the app: which kinds of people may use which features, and whose records may each one see or change? In the standard's words: The authorization rules: which kinds of users may use which functions (function-level: only administrators may open the admin page), and which records each may see or change (data-specific: a person may only open their own notes). **Where to look:** List the kinds of user the app has, then walk through the app as each one and note what they can reach. The pages that only some people should see, and the records that belong to one person, are what these rules are about. **In the code**, comparisons and queries of a record's owner or user id, where the app may keep a person to their own records; places that refuse a request with a 403 (Forbidden), whether the answer is sent, returned, or raised: app.py, line 245; app.py, line 213; app.py, line 320; and 1 more. These are where to look, not evidence: StackVet found them by their names or their shape (a 403 sent, a query by owner) and did not read what they do.
V8.1.2 — Who may see or change each field	tell your AI coding tool your answer, and it writes it in security-notes.md (at a terminal, `sv notes` makes the file)	Where some parts of a record are more private than the rest: who may see each part, who may change it, and does that change once the record is, say, published? In the standard's words: Where some fields of a record are more restricted than the record itself, the field-level access rules: who may read and who may write each field, and whether that depends on the record's state or status (a published order's total can no longer be changed). **Where to look:** Take a record with a field not everybody should see — a price, a note, an internal status. Write down who may read it and who may change it, and whether that depends on the record's state.

requirement	what to do	how
V11.1.1 — How keys are looked after	tell your AI coding tool your answer, and it writes it in security-notes.md (at a terminal, `sv notes` makes the file)	For each secret key the app uses: where is it kept, who can get to it, how often is it replaced, and what happens if it may have leaked? In the standard's words: The policy for cryptographic keys: where each key is kept, who can reach it, how often it is replaced, and what happens when one may have leaked, following a key management standard such as NIST SP 800-57 through each key's lifecycle. **Where to look:** List every key the app uses: session signing, encryption at rest, the TLS certificate, anything a provider gave you. For each, write where it lives, who can read it, and what you would do tomorrow if it leaked.
V11.1.2 — Every key, algorithm, and certificate	tell your AI coding tool your answer, and it writes it in security-notes.md (at a terminal, `sv notes` makes the file)	What is every key, kind of encryption, and certificate the app uses, and what may each one be used for? In the standard's words: A cryptographic inventory: every key, algorithm, and certificate the app uses, where each key may and may not be used, and what data each may and may not protect. Keep it updated when one is added or replaced. **Where to look:** Look in your .env and your host's secret store, at the certificate on the live site, and at whatever the code calls to hash or encrypt. That is the inventory; write down which data each protects and where it may not be used.
V13.1.1 — Everything the app talks to	tell your AI coding tool your answer, and it writes it in security-notes.md (at a terminal, `sv notes` makes the file)	What does the app connect to outside itself (payments, email, AI services, other programs), and why? And can anyone using the app give it an address that it will then visit itself? In the standard's words: All the communication needs of the app: every external service it connects to (payment providers, email, AI services, other APIs) and why, and any place where someone using the app can give it an address that the app will then connect to itself. **Where to look:** Look at your dependency list for anything that calls out — payment, email, AI, maps, analytics — and at the report's own list of outside services. Then add the places where somebody using the app can give it an address it will fetch.
V14.1.2 — How each kind of sensitive data is protected	tell your AI coding tool your answer, and it writes it in security-notes.md (at a terminal, `sv notes` makes the file)	For each kind of private information the app keeps: is it encrypted, how long is it kept, may it appear in logs, who can see it, and what does the privacy law you follow require? In the standard's words: For each kind of sensitive data the app holds, its protection requirements: whether it is encrypted, how long it is kept (retention), whether and how it may appear in logs, who can see it, and anything the privacy law you follow requires. **Where to look:** Take each kind of data from the list you made for V14.1.1 and decide four things for it: encrypted or not, kept for how long, allowed in logs or not, and who may see it.
V15.1.1 — How quickly outdated libraries are updated	tell your AI coding tool your answer, and it writes it in security-notes.md (at a terminal, `sv notes` makes the file)	When a library the app uses turns out to have a security hole, how soon must it be updated, by how serious the hole is (critical within a week, say)? And how often are libraries updated anyway? In the standard's words: Time frames for remediation: how soon a third-party library with a known vulnerability is updated, by how serious the vulnerability is (for example, critical within a week), and how often libraries are updated in general. **Where to look:** Decide the number of days you would accept between a vulnerability being published and it being fixed, for each level of severity. Start from how bad it would be for this app rather than from what other people write down. Then put the same numbers in stackvet.toml as `[policy] fix-within-days`, so `sv audit` can hold the app's packages to them.

requirement	what to do	how
V15.1.3 — Slow or heavy features	tell your AI coding tool your answer, and it writes it in security-notes.md (at a terminal, `sv notes` makes the file)	Which features are slow or heavy (big exports, reports, AI requests, image processing), and what stops one person's heavy use from making the app unavailable to everyone else? In the standard's words: Which functionality is time-consuming or resource-demanding (large exports, report generation, AI requests, image processing), and how the app prevents a loss of availability from overusing it: queues, limits per user, timeouts. **Where to look:** Time the slow things: an export, a report, a search over everything, an AI request, anything that resizes an image. Anything a person notices waiting for is what this asks about. **In the code**, work that can be slow or heavy: password hashing, images and PDFs, archives, exports, calls to an AI model, and files uploaded: app.py, line 49. These are where to look, not evidence: StackVet found them by their names or their shape (a 403 sent, a query by owner) and did not read what they do.
V16.1.1 — What the app logs	tell your AI coding tool your answer, and it writes it in security-notes.md (at a terminal, `sv notes` makes the file)	What does the app write in its logs, where are they kept, who can read them, what are they for, and how long are they kept? In the standard's words: An inventory of the logging: which events are logged, in what format, where the logs are stored, who can read them (access), what they are used for, and how long they are kept. **Where to look:** Find where the app's output actually goes — the console, a file, your host's logging, an outside service — and what it writes at each level. Then say who can read it and how long it is kept.
V8.3.1 — Authorization rules are enforced at a trusted service layer	tell your AI coding tool your answer, and it records it in the `[design]` section of stackvet.toml	When somebody tries to open a page or do something they are not allowed to, is it the server that stops them, or only a button that is hidden? In the standard's words: Does the application enforce its authorization rules at a trusted service layer — the server — rather than relying on controls an untrusted consumer could manipulate, such as a hidden button or a disabled field in the browser? Hiding a link is not a rule: anyone can request the address directly. **Where to look:** Take a page or an action only some people should reach, and find the code that stops everybody else. If the only thing you find is the button being hidden in the template, that is the answer. **In the code**, comparisons and queries of a record's owner or user id, where the app may keep a person to their own records; places that refuse a request with a 403 (Forbidden), whether the answer is sent, returned, or raised: app.py, line 245; app.py, line 213; app.py, line 320; and 1 more. These are where to look, not evidence: StackVet found them by their names or their shape (a 403 sent, a query by owner) and did not read what they do.
V4.2.1 — Everything in front of the app agrees where a request ends	tell your AI coding tool your answer, and it records it in the `[design]` section of stackvet.toml	If a request passes through other machines before reaching the app (a load balancer, a firewall), do they all agree on where one request ends and the next begins? In the standard's words: Do all the components a request passes through — load balancers, firewalls, and the application server — agree on where one HTTP message ends and the next begins? When they disagree, somebody can hide a second request inside the first. **Where to look:** Find what sits in front of the app — a load balancer, a reverse proxy, your host's edge. Check whether each one and the app agree on request size limits, chunked encoding, and duplicate headers. If you run nothing in front of the app, say so: that is the answer. **Why StackVet cannot check it:** The machines in front of the app, such as a load balancer, a reverse proxy, or your host's edge, are not in the app's folder, and StackVet runs the app on its own, with nothing in front of it. Sending the app a request built to be read two ways would test only the app's own server, and would say nothing about what is in front of it once the app is live.

requirement	what to do	how
V14.2.2 — Sensitive data is not left in a cache	tell your AI coding tool your answer, and it records it in the `[design]` section of stackvet.toml	Is private information kept out of the caches the app does not control (your host's edge, a content delivery network), or protected when it is stored there? In the standard's words: Does the application prevent sensitive data from being kept in caches it does not own — a load balancer, a content delivery network, or an application cache — or make sure anything kept there is protected? **Where to look:** List every cache a response passes through: your host's edge or CDN, a reverse proxy, the browser. Then look at what a private page sends back in Cache-Control. Anything sensitive without no-store is the answer to this one.
V16.4.3 — Logs are sent somewhere separate	tell your AI coding tool your answer, and it records it in the `[design]` section of stackvet.toml	Are the app's logs sent to a separate place, so someone who took over the app could not quietly delete them? In the standard's words: Are logs sent to a separate system, so that somebody who takes over the application cannot quietly delete the record of what they did? **Where to look:** Find where the logs end up. If the answer is a file on the same machine the app runs on, then somebody who takes over the app can edit them, and this is not done.
V2.3.1 — Steps cannot be skipped or done out of order	check it by hand; nothing here can	Take a flow with more than one step — checkout, sign-up with a confirmation, anything with a "next" button. Write down the address of the last step. Start the flow again, and go straight to that address without doing the steps before it. It should refuse. Then try the steps in the wrong order, and try the same step twice. Describing the flow in stackvet.toml as `flow` under `[stack.run.users]` lets `sv report --run` try two skips, one wrong order, and one repeated step for you; other orders, and the last step sent again after the flow finished, are still yours.
V12.2.2 — The certificate on the live site is one browsers trust	check it by hand; nothing here can	Open the live site in a browser and click the padlock. It should show a certificate issued by a public authority, with a date that has not passed, and the name should match the address. A warning page, or a certificate the app generated for itself, means no visitor can tell your site from an impostor.
V15.2.1 — Nothing is past the update deadline you set	check it by hand; nothing here can	Write the time frames from your security notes into stackvet.toml as `[policy] fix-within-days`, one number of days per severity, then run `sv report --advisories DIR` (or `sv audit`) against a local advisory database. It lists the packages with a known vulnerability and puts the ones past your time frame first. Those are the answer to this one. Without the numbers it counts every known vulnerability as late, whatever its age.
V2.3.2 — The business limits you wrote down are the ones the app enforces	check it by hand; nothing here can	Take each limit from your security notes — so many of something per person, so many per day — and try to go past it in the app. Try it slowly and then quickly, and from two accounts at once if the limit is meant to be global.
V2.3.4 — Two people cannot book the same thing	check it by hand; nothing here can	Only where something is limited in number: a seat, a slot, the last item in stock. Open the app in two browsers, get both to the moment of booking the same one, and confirm both as close together as you can. Exactly one should succeed.
V6.2.11 — The banned word list is actually used	check it by hand; nothing here can	Take a word from the list you wrote down — the app's name, your organization's name — and try to register an account using it as the password, or with it inside a longer password. It should be refused. Listing the words in stackvet.toml under `[policy]` as `context-words` lets `sv report --run` try the first one for you, through the app's own sign-up.
V6.2.12 — Passwords from known breaches are refused	check it by hand; nothing here can	Try to register with a password known to have been breached. `Password123!` and `qwerty123456` will do; haveibeenpwned.com/Passwords lists more. The app should refuse it and say why, without telling the person anything about other accounts. `sv report --run` tries one such password through the app's own sign-up, but one refusal only supports this: it does not show the whole list is checked.

requirement	what to do	how
V6.3.3 — Getting in needs more than a password	check it by hand; nothing here can	Sign in and note everything you were asked for. A password alone is a single factor. A code from an app or a security key is a second one. If there is only a password, that is the answer to this one; if you have decided a password alone is enough, ASVS asks for the reasoning to be written down.
V6.3.4 — There is no forgotten way in	check it by hand; nothing here can	List every way somebody can become signed in: the password form, any "sign in with" button, magic links by email, API keys, an admin back door, an old mobile app. Compare that with the list in your security notes. Anything not on the list is what this asks about, and each way in should be as strong as the others.
V6.4.4 — Losing a second factor does not become the easy way in	check it by hand; nothing here can	Only where people sign in with a second factor. Go through losing it as a person would: what does support ask for before restoring access? If the answer is less than what was asked when the account was set up, the recovery path is now the weakest way in.
V7.3.1 — Being away signs you out	check it by hand; nothing here can	Sign in, then leave the app alone for longer than the idle timeout you wrote down, without clicking anything. Come back and reload a private page. You should be asked to sign in again.
V7.3.2 — A session does not last forever	check it by hand; nothing here can	Sign in and keep using the app past the maximum lifetime you wrote down. Staying active must not keep it alive forever — at the limit you should be asked to sign in again even mid-use.
V13.3.1 — Secrets live in something built to hold them	check it by hand; nothing here can	List where each password, key, and token the app needs actually comes from: a key vault or your host's secret store, an environment variable, or a file in the code. Say who can read each one, and how you would replace one if it leaked. A secret nobody can rotate is the answer to this one.
V13.4.5 — Nothing internal is reachable from outside	check it by hand; nothing here can	From a browser that is not signed in, and ideally not on your network, try the addresses that exist for you rather than for visitors: /docs, /swagger, /metrics, /health, /debug, /admin, /.git. Each should either refuse or not exist.
V14.1.1 — You know which of your data is sensitive	check it by hand; nothing here can	List every kind of information the app stores, including what it writes to logs and what it keeps in backups. Mark each with how much it would matter if it got out. Anything you cannot place is the thing to look at, and remember that encoded is not the same as protected.
V14.2.3 — Nothing sensitive is handed to an outside service	check it by hand; nothing here can	Open the app in a browser with the network panel recording, sign in, and use it normally. Look at every request going somewhere that is not your own address — analytics, error reporting, fonts, chat widgets. Check what is in each, including the address of the page, which often carries names or record numbers. With `browser` under `[stack.run.users]`, `sv report --run` looks through what the signed-in pages send to other sites for the test account's email address, password, and session cookie; but code a page loads from another site cannot run inside the fence, so what analytics and chat widgets send is still yours to look at.
V14.2.4 — Sensitive data is handled the way your notes say	check it by hand; nothing here can	Take each protection you wrote down — encrypted, kept for so long, never in logs — and check the app against it. Search the logs for the data that should never appear there, and confirm that something really deletes records when their time is up.
V15.2.2 — Time-consuming features cannot cause a loss of availability	check it by hand; nothing here can	Take each time-consuming or resource-demanding feature you documented — an export, a report, an AI request, image processing. Start several at once from one account and see whether the rest of the application stays usable. The defenses you wrote down should stop any one person causing a loss of availability for everybody else. **In the code**, work that can be slow or heavy: password hashing, images and PDFs, archives, exports, calls to an AI model, and files uploaded: app.py, line 49. These are where to look, not evidence: StackVet found them by their names or their shape (a 403 sent, a query by owner) and did not read what they do.

requirement	what to do	how
V16.2.3 — Logs go only where your inventory says	check it by hand; nothing here can	Compare the log destinations in your inventory with where they really go: files on the server, your host's logging, any outside service, and anything printed to the console. A destination nobody wrote down is one nobody is watching or protecting.
V16.3.3 — The events you said would be logged really are	check it by hand; nothing here can	Do each thing your notes say gets logged — a failed sign-in, a permission refused, a record deleted — then look in the logs and confirm each one is there with enough detail to tell who and when. Then try something that should be refused, such as posting a form with a field the app does not expect, and check that the refusal was logged too.
V5.4.3 — Uploaded files are scanned for viruses	check it by hand; nothing here can	Find where a file someone uploads is stored before anyone downloads it, and check whether it is scanned by antivirus software on the way in: your host's upload service may offer this, or a scanner such as ClamAV can be added. Try uploading the harmless EICAR test file, which every antivirus scanner recognizes. If the app accepts it and serves it back, known malicious content would be served too.
C7.2.1 — The app judges how reliable each AI answer is	check it by hand; nothing here can	Ask your AI coding tool to show you where the app decides how confident it is in an answer the model generated, before showing it. Look for a score or a check, not just the model's text passed straight through. If the answer is shown whatever the model said, the reliability of generated answers is not being assessed.
C7.2.2 — A doubtful AI answer is blocked or replaced	check it by hand; nothing here can	With the confidence check from the item above, find the threshold below which an answer is not shown, and what the app shows instead: a fallback message such as "I'm not sure, please contact us". Try asking the AI something it cannot know about your business. It should fall back rather than invent an answer.
C12.2.2 — Unusual behavior toward the AI is spotted	check it by hand; nothing here can	Check whether the app watches for conversations that look like probing: the same question retried many times with small changes, excessive retry attempts, or one visitor trying many different tricks in a row. Try it yourself and look for whether anything was flagged or anyone told.
C12.2.3 — Jailbreak and prompt-injection attempts are recognized	check it by hand; nothing here can	List the rules the app uses to recognize attacks aimed at the AI: jailbreak phrasings, prompt injection hidden in documents it reads, attempts to extract its system prompt. Try a few coordinated attempts from two accounts at once, and check whether the rules catch them and something is recorded.
AC.4.1 — A person reviews the code the AI wrote	check it by hand; nothing here can	Look at how changes to the app get in: check that every change your AI coding tool wrote is read and approved by a qualified person before it goes live, and that the reviewer is not the same person who asked the AI for the code. The AI agent does not count as the human reviewer. If changes go straight from the AI to the live app, the answer is no.
AC.6.3 — The AI coding tools themselves are attacked on purpose, on a schedule	check it by hand; nothing here can	Check whether anyone regularly tries to trick the AI tools that help build the app: a pull request, issue, or comment with a hidden instruction, a known jailbreak, or a request to add a harmful package. Write down when this was last done and what it found, and whether each finding was tracked until fixed. If it has never been done, schedule the first red-team exercise.

Before going live

These can be answered only by the app's own live site, so nothing in this report has checked them, and none is counted as met. Once the app is on the internet at its own address, run `sv probe` against it: it makes a handful of read-only requests to that address and nothing else, sends no cookies or credentials, and cannot change anything. It prints its answers where it runs; they are not added to this report.

requirement	what it asks	how
V12.2.1	The site answers over HTTPS, and plain HTTP is no way in	`sv probe https://your-address` asks this.
V12.2.2	Its certificate is one browsers trust	`sv probe https://your-address` asks this.
V12.1.1	Old versions of TLS (1.0 and 1.1) are refused	`sv probe https://your-address` asks this.
V3.4.1	Browsers are told to use HTTPS only (HSTS)	`sv probe https://your-address` asks this.
V3.3.3	The session cookie's name carries the `__Host-` prefix	`sv probe https://your-address` asks this.
V4.1.2	A program calling the API over plain HTTP is refused rather than redirected	`sv probe https://your-address --api /a/path/of/your/api` asks this.
V12.1.2	Only strong encryption settings (cipher suites) are offered	Not sv probe: check it with a scanner made for it, such as testssl.sh or SSL Labs' online test (see "What only you can check").

What the AI coding tool did that is worth a look

Information only: nothing here changes a requirement's status, counts toward anything, or is a finding. Each line is something the AI coding tool did that a person may want to look at, with when and where it was seen. Many of them are right to do; a line says what happened, not that it was wrong.

Read: the last 20 commits, as there is no earlier report made at a known commit, of which 20 were made by an AI coding tool, by a trailer in its message that names one; the 0 that name none are not read as the tool's.

Not looked for: the commands the tool ran and the files it wrote without committing them, since the hook record was not read for this run (`sv report --hook-record` reads it, once its hook is installed).

Not looked for: findings that went away, since this folder held no earlier report that sv can show it wrote.

Kind	What was seen	Where
A guard edited	.github/workflows/rust.yml was changed	commit afca5627 "Every file that decides what the required check runs is code", by Claude Opus 5.5 <noreply@anthropic.com>

Tests worth writing first

Nothing produced evidence about these, and no test in the app names them. A test that names a requirement's id and passes is the one way to give evidence about any requirement, including the ones no check here can reach. These are the level 1 ones. Name only what a test really checks: nothing here can tell whether it does.

73 more are at level 2 and above. They are not listed here, because a list that long is not something a person works through; `sv mcp` gives the whole list to your AI coding tool, and report.json carries it under `tests_to_write`.

102 more have no evidence and are not listed at all, because an application's own tests cannot show them: they ask for documentation, a deployment setting, a development process, or a design decision, and a person answers them.

requirement	what it asks for

requirement	what it asks for
V1.2.1	Verify that output encoding for an HTTP response, HTML document, or XML document is relevant for the context required, such as encoding the relevant characters for HTML elements, HTML attributes, HTML comments, CSS, or HTTP header fields, to avoid changing the message or document structure.
V1.2.2	Verify that when dynamically building URLs, untrusted data is encoded according to its context (e.g., URL encoding or base64url encoding for query or path parameters). Ensure that only safe URL protocols are permitted (e.g., disallow javascript: or data:).
V1.2.3	Verify that output encoding or escaping is used when dynamically building JavaScript content (including JSON), to avoid changing the message or document structure (to avoid JavaScript and JSON injection).
V2.2.1	Verify that input is validated to enforce business or functional expectations for that input. This should either use positive validation against an allow list of values, patterns, and ranges, or be based on comparing the input to an expected structure and logical limits according to predefined rules. For L1, this can focus on input which is used to make specific business or security decisions. For L2 and up, this should apply to all input.
V2.2.2	Verify that the application is designed to enforce input validation at a trusted service layer. While client-side validation improves usability and should be encouraged, it must not be relied upon as a security control.
V3.2.1	Verify that security controls are in place to prevent browsers from rendering content or functionality in HTTP responses in an incorrect context (e.g., when an API, a user-uploaded file or other resource is requested directly). Possible controls could include: not serving the content unless HTTP request header fields (such as Sec-Fetch-\) *indicate it is the correct context, using the sandbox directive of the Content-Security-Policy header field or using the attachment disposition type in the Content-Disposition header field.*
V3.2.2	Verify that content intended to be displayed as text, rather than rendered as HTML, is handled using safe rendering functions (such as createTextNode or textContent) to prevent unintended execution of content such as HTML or JavaScript.
V3.3.1	Verify that cookies have the 'Secure' attribute set, and if the '__Host-' prefix is not used for the cookie name, the '__Secure-' prefix must be used for the cookie name.
V3.4.1	Verify that a Strict-Transport-Security header field is included on all responses to enforce an HTTP Strict Transport Security (HSTS) policy. A maximum age of at least 1 year must be defined, and for L2 and up, the policy must apply to all subdomains as well.
V3.4.2	Verify that the Cross-Origin Resource Sharing (CORS) Access-Control-Allow-Origin header field is a fixed value by the application, or if the Origin HTTP request header field value is used, it is validated against an allowlist of trusted origins. When 'Access-Control-Allow-Origin: *' needs to be used, verify that the response does not include any sensitive information.*
V3.5.1	Verify that, if the application does not rely on the CORS preflight mechanism to prevent disallowed cross-origin requests to use sensitive functionality, these requests are validated to ensure they originate from the application itself. This may be done by using and validating anti-forgery tokens or requiring extra HTTP header fields that are not CORS-safelisted request-header fields. This is to defend against browser-based request forgery attacks, commonly known as cross-site request forgery (CSRF).
V3.5.2	Verify that, if the application relies on the CORS preflight mechanism to prevent disallowed cross-origin use of sensitive functionality, it is not possible to call the functionality with a request which does not trigger a CORS-preflight request. This may require checking the values of the 'Origin' and 'Content-Type' request header fields or using an extra header field that is not a CORS-safelisted header-field.
V3.5.3	Verify that HTTP requests to sensitive functionality use appropriate HTTP methods such as POST, PUT, PATCH, or DELETE, and not methods defined by the HTTP specification as "safe" such as HEAD, OPTIONS, or GET. Alternatively, strict validation of the Sec-Fetch-* request header fields can be used to ensure that the request did not originate from an inappropriate cross-origin call, a navigation request, or a resource load (such as an image source) where this is not expected.
V4.1.1	Verify that every HTTP response with a message body contains a Content-Type header field that matches the actual content of the response, including the charset parameter to specify safe character encoding (e.g., UTF-8, ISO-8859-1) according to IANA Media Types, such as "text/", "/+xml" and "/xml".

requirement	what it asks for
V5.2.1	Verify that the application will only accept files of a size which it can process without causing a loss of performance or a denial of service attack.
V5.2.2	Verify that when the application accepts a file, either on its own or within an archive such as a zip file, it checks if the file extension matches an expected file extension and validates that the contents correspond to the type represented by the extension. This includes, but is not limited to, checking the initial 'magic bytes', performing image re-writing, and using specialized libraries for file content validation. For L1, this can focus just on files which are used to make specific business or security decisions. For L2 and up, this must apply to all files being accepted.
V5.3.1	Verify that files uploaded or generated by untrusted input and stored in a public folder, are not executed as server-side program code when accessed directly with an HTTP request.
V6.2.1	Verify that user set passwords are at least 8 characters in length although a minimum of 15 characters is strongly recommended.
V6.2.2	Verify that users can change their password.
V6.2.3	Verify that password change functionality requires the user's current and new password.
V6.2.4	Verify that passwords submitted during account registration or password change are checked against an available set of, at least, the top 3000 passwords which match the application's password policy, e.g. minimum length.
V6.2.5	Verify that passwords of any composition can be used, without rules limiting the type of characters permitted. There must be no requirement for a minimum number of upper or lower case characters, numbers, or special characters.
V6.2.6	Verify that password input fields use type=password to mask the entry. Applications may allow the user to temporarily view the entire masked password, or the last typed character of the password.
V6.2.7	Verify that "paste" functionality, browser password helpers, and external password managers are permitted.
V6.2.8	Verify that the application verifies the user's password exactly as received from the user, without any modifications such as truncation or case transformation.
V6.3.2	Verify that default user accounts (e.g., "root", "admin", or "sa") are not present in the application or are disabled.
V6.4.1	Verify that system generated initial passwords or activation codes are securely randomly generated, follow the existing password policy, and expire after a short period of time or after they are initially used. These initial secrets must not be permitted to become the long term password.
V6.4.2	Verify that password hints or knowledge-based authentication (so-called "secret questions") are not present.
V7.2.1	Verify that the application performs all session token verification using a trusted, backend service.
V7.2.2	Verify that the application uses either self-contained or reference tokens that are dynamically generated for session management, i.e. not using static API secrets and keys.
V7.2.3	Verify that if reference tokens are used to represent user sessions, they are unique and generated using a cryptographically secure pseudo-random number generator (CSPRNG) and possess at least 128 bits of entropy.
V7.2.4	Verify that the application generates a new session token on user authentication, including re-authentication, and terminates the current session token.
V7.4.1	Verify that when session termination is triggered (such as logout or expiration), the application disallows any further use of the session. For reference tokens or stateful sessions, this means invalidating the session data at the application backend. Applications using self-contained tokens will need a solution such as maintaining a list of terminated tokens, disallowing tokens produced before a per-user date and time or rotating a per-user signing key.
V7.4.2	Verify that the application terminates all active sessions when a user account is disabled or deleted (such as an employee leaving the company).
V8.2.1	Verify that the application ensures that function-level access is restricted to consumers with explicit permissions.
V8.2.2	Verify that the application ensures that data-specific access is restricted to consumers with explicit permissions to specific data items to mitigate insecure direct object reference (IDOR) and broken object level authorization (BOLA).
V11.3.1	Verify that insecure block modes (e.g., ECB) and weak padding schemes (e.g., PKCS#1 v1.5) are not used.

requirement	what it asks for
V12.1.1	Verify that only the latest recommended versions of the TLS protocol are enabled, such as TLS 1.2 and TLS 1.3. The latest version of the TLS protocol must be the preferred option.
V12.2.1	Verify that TLS is used for all connectivity between a client and external facing, HTTP-based services, and does not fall back to insecure or unencrypted communications.
V13.4.1	Verify that the application is deployed either without any source control metadata, including the .git or .svn folders, or in a way that these folders are inaccessible both externally and to the application itself.
V14.2.1	Verify that sensitive data is only sent to the server in the HTTP message body or header fields, and that the URL and query string do not contain sensitive information, such as an API key or session token.
V14.3.1	Verify that authenticated data is cleared from client storage, such as the browser DOM, after the client or session is terminated. The 'Clear-Site-Data' HTTP response header field may be able to help with this but the client-side should also be able to clear up if the server connection is not available when the session is terminated.
V15.3.1	Verify that the application only returns the required subset of fields from a data object. For example, it should not return an entire data object, as some individual fields should not be accessible to users.

Requirements nobody has placed — each names the question that would place it

These are not exclusions. Answering the question in the second column moves each one into *applies* or *does not apply*.

requirement	what has to be answered
AC.11.7	Does the app take code contributions from outside people?
AC.12.1	Does the app have a CI/CD pipeline?
AC.12.2	Does the app have a CI/CD pipeline?
AC.12.3	Does the app have a CI/CD pipeline?
AC.12.4	Does the app have a CI/CD pipeline?
AC.12.5	Does the app have a CI/CD pipeline?
AC.12.6	Does the app have a CI/CD pipeline?
AC.12.7	Does the app have a CI/CD pipeline?
AC.12.8	Does the app have a CI/CD pipeline?
AC.13.1	Does the app take code contributions from outside people?
AC.13.2	Does the app take code contributions from outside people?
AC.13.3	Does the app take code contributions from outside people?
AC.13.4	Does the app take code contributions from outside people?
AC.6.1	Does the app train or fine-tune an AI model?
AC.6.2	Does the app train or fine-tune an AI model?
AC.7.3	Does the app have infrastructure or pipeline configuration (Terraform, CloudFormation, CI workflows)?
AC.7.4	Does the app have a CI/CD pipeline?
AC.8.3	Is the app's code on a source-control server with branch protection or a merge queue?
AC.9.1	Does the app have a CI/CD pipeline?
C1.1.1	Does the app train or fine-tune an AI model?
C1.1.2	Does the app train or fine-tune an AI model?

requirement	what has to be answered
C1.1.3	Does the app train or fine-tune an AI model?
C1.1.4	Does the app train or fine-tune an AI model?
C1.2.1	Does the app train or fine-tune an AI model?
C1.2.2	Does the app train or fine-tune an AI model?
C1.2.3	Does the app train or fine-tune an AI model?
C1.3.1	Does the app train or fine-tune an AI model?
C1.3.2	Does the app train or fine-tune an AI model?
C1.3.3	Does the app train or fine-tune an AI model?
C1.3.4	Does the app train or fine-tune an AI model?
C10.1.1	Does the AI use the Model Context Protocol (MCP) to talk to tools?
C10.1.2	Does the AI use the Model Context Protocol (MCP) to talk to tools?
C10.1.3	Does the AI use the Model Context Protocol (MCP) to talk to tools?
C10.2.1	Does the app serve tools to AI models over the Model Context Protocol (MCP)?
C10.2.2	Does the app serve tools to AI models over the Model Context Protocol (MCP)?
C10.2.3	Does the app serve tools to AI models over the Model Context Protocol (MCP)?
C10.2.4	Does the app serve tools to AI models over the Model Context Protocol (MCP)?
C10.2.5	Does the app serve tools to AI models over the Model Context Protocol (MCP)?
C10.2.6	Does the app serve tools to AI models over the Model Context Protocol (MCP)?
C10.2.7	Does the app serve tools to AI models over the Model Context Protocol (MCP)?
C10.3.1	Does the AI use the Model Context Protocol (MCP) to talk to tools? — or — Does the app serve tools to AI models over the Model Context Protocol (MCP)?
C10.3.2	Does the AI use the Model Context Protocol (MCP) to talk to tools? — or — Does the app serve tools to AI models over the Model Context Protocol (MCP)?
C10.3.3	Does the app serve tools to AI models over the Model Context Protocol (MCP)?
C10.3.4	Does the AI use the Model Context Protocol (MCP) to talk to tools?
C10.4.1	Does the AI use the Model Context Protocol (MCP) to talk to tools?
C10.4.2	Does the AI use the Model Context Protocol (MCP) to talk to tools?
C10.4.3	Does the app serve tools to AI models over the Model Context Protocol (MCP)?
C10.4.4	Does the app serve tools to AI models over the Model Context Protocol (MCP)?
C10.4.5	Does the AI use the Model Context Protocol (MCP) to talk to tools? — or — Does the app serve tools to AI models over the Model Context Protocol (MCP)?
C10.4.6	Does the app serve tools to AI models over the Model Context Protocol (MCP)?
C10.4.7	Does the AI use the Model Context Protocol (MCP) to talk to tools?
C11.1.1	Does the app have AI features?
C11.1.2	Does the app have AI features?
C11.1.3	Does the app have AI features?
C11.1.4	Does the app have AI features?

requirement	what has to be answered
C11.2.1	Does the app have AI features?
C11.2.2	Does the app have AI features?
C11.2.3	Does the app host, build, or deploy AI models of its own?
C11.2.4	Does the app train or fine-tune an AI model?
C11.3.1	Does the app have AI features?
C11.3.2	Does the app have AI features?
C11.4.1	Is content moderation turned on for the AI?
C11.4.2	Is content moderation turned on for the AI?
C12.1.1	Does the app have AI features?
C12.1.2	Does the app have AI features?
C12.1.3	Does the app have AI features?
C12.1.4	Does the AI search a document store or vector database (retrieval-augmented generation)? — or — Does the AI search the web or read web pages?
C12.2.1	Does the app have AI features?
C12.2.5	Does the app have AI features?
C12.3.1	Does the app host, build, or deploy AI models of its own?
C12.3.2	Does the app host, build, or deploy AI models of its own?
C12.3.3	Does the app host, build, or deploy AI models of its own?
C12.4.1	Can the AI change data or take actions, rather than only answer questions?
C12.4.2	Can the AI change data or take actions, rather than only answer questions?
C12.4.3	Does the app have AI features?
C12.5.1	Does the app train or fine-tune an AI model?
C12.5.2	Does the app train or fine-tune an AI model?
C12.5.3	Does the app train or fine-tune an AI model?
C12.5.4	Does the app train or fine-tune an AI model?
C2.1.1	Does the app have AI features?
C2.1.2	Does the app have AI features?
C2.1.3	Does the app have AI features?
C2.1.4	Does the app have AI features?
C2.1.5	Does the app have AI features?
C2.1.6	Does the app have AI features?
C2.1.7	Does the app have AI features?
C2.2.1	Is content moderation turned on for the AI?
C2.2.2	Is content moderation turned on for the AI?
C2.2.3	Does the assistant take images, video, or audio, not only typed text?
C3.1.1	Does the app have AI features?
C3.1.2	Does the app host, build, or deploy AI models of its own?

requirement	what has to be answered
C3.1.3	Does the app host, build, or deploy AI models of its own?
C3.2.1	Does the app have AI features?
C3.2.2	Does the app host, build, or deploy AI models of its own?
C3.3.1	Does the app host, build, or deploy AI models of its own?
C3.3.2	Does the app host, build, or deploy AI models of its own?
C3.3.3	Does the app host, build, or deploy AI models of its own?
C3.4.1	Does the app host, build, or deploy AI models of its own?
C3.4.2	Does the app host, build, or deploy AI models of its own?
C3.5.1	Does the app host, build, or deploy AI models of its own?
C4.1.1	Does the app host, build, or deploy AI models of its own?
C4.1.2	Does the app host, build, or deploy AI models of its own?
C4.2.1	Does the app host, build, or deploy AI models of its own?
C4.3.1	Does the app host, build, or deploy AI models of its own?
C4.3.2	Does the app host, build, or deploy AI models of its own?
C5.2.1	Does the app have AI features?
C5.2.2	Does the AI search a document store or vector database (retrieval-augmented generation)?
C5.2.3	Does the AI search a document store or vector database (retrieval-augmented generation)?
C5.2.4	Does the app have AI features?
C5.2.5	Can the AI change data or take actions, rather than only answer questions?
C5.3.1	Does the app serve several separate customer organizations that share one system?
C6.1.1	Does the app host, build, or deploy AI models of its own?
C6.1.2	Does the app have AI features?
C6.1.3	Does the app host, build, or deploy AI models of its own?
C6.1.4	Does the app host, build, or deploy AI models of its own?
C6.2.1	Does the app have AI features?
C6.2.2	Does the app host, build, or deploy AI models of its own?
C6.2.3	Does the app host, build, or deploy AI models of its own?
C7.1.1	Does the app have AI features?
C7.1.2	Does the app have AI features?
C7.3.1	Is content moderation turned on for the AI?
C7.3.2	Does the app have AI features?
C7.3.3	Does the app have AI features?
C7.4.1	Does the AI search a document store or vector database (retrieval-augmented generation)? — or — Does the AI search the web or read web pages?
C7.4.2	Does the AI search a document store or vector database (retrieval-augmented generation)? — or — Does the AI search the web or read web pages?

requirement	what has to be answered
C7.4.3	Does the AI search a document store or vector database (retrieval-augmented generation)? — or — Does the AI search the web or read web pages?
C8.1.1	Does the AI search a document store or vector database (retrieval-augmented generation)?
C8.1.2	Does the AI search a document store or vector database (retrieval-augmented generation)?
C8.1.3	Does the AI keep conversation history between visits?
C8.2.1	Does the AI search a document store or vector database (retrieval-augmented generation)?
C8.2.2	Does the AI search a document store or vector database (retrieval-augmented generation)?
C8.2.3	Does the AI search a document store or vector database (retrieval-augmented generation)?
C8.3.1	Does the AI keep conversation history between visits?
C8.3.2	Does the AI keep conversation history between visits?
C9.1.1	Does the app have AI features?
C9.1.2	Does the app have AI features?
C9.1.3	Does the app have AI features?
C9.2.1	Can the AI change data or take actions, rather than only answer questions?
C9.2.2	Can the AI change data or take actions, rather than only answer questions?
C9.2.3	Can the AI change data or take actions, rather than only answer questions?
C9.2.4	Can the AI change data or take actions, rather than only answer questions?
C9.2.5	Can the AI change data or take actions, rather than only answer questions?
C9.2.6	Can the AI change data or take actions, rather than only answer questions?
C9.2.7	Can the AI change data or take actions, rather than only answer questions?
C9.3.1	Can the AI change data or take actions, rather than only answer questions?
C9.3.2	Can the AI change data or take actions, rather than only answer questions?
C9.3.3	Can the AI change data or take actions, rather than only answer questions?
C9.3.4	Can the AI change data or take actions, rather than only answer questions?
C9.3.5	Can the AI change data or take actions, rather than only answer questions?
C9.3.6	Can the AI change data or take actions, rather than only answer questions?
C9.3.7	Can the AI change data or take actions, rather than only answer questions?
C9.4.1	Are there several AI agents that must identify each other, rather than a single assistant?
C9.4.2	Are there several AI agents that must identify each other, rather than a single assistant?
C9.5.1	Can the AI change data or take actions, rather than only answer questions?
C9.5.2	Can the AI change data or take actions, rather than only answer questions?
C9.5.3	Does the app have AI features?
C9.5.4	Does the app have AI features?
C9.5.5	Can the AI change data or take actions, rather than only answer questions?
C9.6.1	Does the app have AI features?
C9.6.2	Does the app have AI features?
SBD-AC-04	Does the app run as more than one service?

requirement	what has to be answered
SBD-AC-07	Does the app have a CI/CD pipeline?
SBD-AS-01	Does the app run as more than one service?
SBD-AS-02	Does the app run as more than one service?
SBD-AS-03	Does the app run as more than one service?
SBD-AS-04	Does the app run as more than one service?
SBD-AS-05	Does the app run as more than one service?
SBD-AS-06	Does the app run as more than one service? — or — Does the app run scheduled background jobs?
SBD-AS-08	Does the app run as more than one service?
SBD-DM-03	Does the app run as more than one service? — or — Does the app take payments? — or — Does the app run scheduled background jobs?
SBD-DM-04	Does the app run as more than one service?
SBD-DM-06	Does the app run as more than one service?
SBD-RR-02	Does the app run as more than one service? — or — Does the app call other services over the network?
SBD-RR-03	Does the app run as more than one service? — or — Does the app run scheduled background jobs?
SBD-RR-04	Does the app run as more than one service?
V1.3.1	Does the app take formatted text that people supply and show it: rich text from an editor, or Markdown, stylesheets, templates, BBCode, or similar? — or — Does the app have AI features?
V1.3.11	Does the app send email?
V1.3.5	Does the app take formatted text that people supply and show it: rich text from an editor, or Markdown, stylesheets, templates, BBCode, or similar? — or — Does the app have AI features?
V10.1.1	Does the app sign people in through another provider (OAuth or OpenID Connect)?
V10.1.2	Does the app sign people in through another provider (OAuth or OpenID Connect)?
V10.2.1	Does the app sign people in through another provider (OAuth or OpenID Connect)?
V10.2.2	Does the app sign people in through another provider (OAuth or OpenID Connect)?
V10.3.1	Does the app sign people in through another provider (OAuth or OpenID Connect)?
V10.3.2	Does the app sign people in through another provider (OAuth or OpenID Connect)?
V10.3.3	Does the app sign people in through another provider (OAuth or OpenID Connect)?
V10.3.4	Does the app sign people in through another provider (OAuth or OpenID Connect)?
V10.4.1	Does the app run an OAuth authorization server or OpenID provider of its own?
V10.4.10	Does the app run an OAuth authorization server or OpenID provider of its own?
V10.4.11	Does the app run an OAuth authorization server or OpenID provider of its own?
V10.4.2	Does the app run an OAuth authorization server or OpenID provider of its own?
V10.4.3	Does the app run an OAuth authorization server or OpenID provider of its own?
V10.4.4	Does the app run an OAuth authorization server or OpenID provider of its own?
V10.4.5	Does the app run an OAuth authorization server or OpenID provider of its own?
V10.4.6	Does the app run an OAuth authorization server or OpenID provider of its own?
V10.4.7	Does the app run an OAuth authorization server or OpenID provider of its own?

requirement	what has to be answered
V10.4.8	Does the app run an OAuth authorization server or OpenID provider of its own?
V10.4.9	Does the app run an OAuth authorization server or OpenID provider of its own?
V10.5.1	Does the app sign people in through another provider (OAuth or OpenID Connect)?
V10.5.2	Does the app sign people in through another provider (OAuth or OpenID Connect)?
V10.5.3	Does the app sign people in through another provider (OAuth or OpenID Connect)?
V10.5.4	Does the app sign people in through another provider (OAuth or OpenID Connect)?
V10.5.5	Does the app sign people in through another provider (OAuth or OpenID Connect)?
V10.6.1	Does the app run an OAuth authorization server or OpenID provider of its own?
V10.6.2	Does the app run an OAuth authorization server or OpenID provider of its own?
V10.7.1	Does the app run an OAuth authorization server or OpenID provider of its own?
V10.7.2	Does the app run an OAuth authorization server or OpenID provider of its own?
V10.7.3	Does the app run an OAuth authorization server or OpenID provider of its own?
V12.3.1	Does the app call other services over the network? — or — Does the app have AI features? — or — Does the app send email?
V12.3.2	Does the app call other services over the network? — or — Does the app have AI features? — or — Does the app send email?
V12.3.3	Does the app call other services over the network? — or — Does the app have AI features? — or — Does the app send email?
V12.3.4	Does the app call other services over the network? — or — Does the app have AI features? — or — Does the app send email?
V13.2.1	Does the app call other services over the network?
V13.2.2	Does the app call other services over the network?
V13.2.4	Does the app call other services over the network? — or — Does the app have AI features? — or — Does the app send email?
V13.2.5	Does the app call other services over the network? — or — Does the app have AI features? — or — Does the app send email?
V15.3.2	Does the app call other services over the network? — or — Does the app have AI features? — or — Does the app send email?
V16.5.2	Does the app call other services over the network? — or — Does the app have AI features? — or — Does the app send email?
V17.1.1	Does the app have real-time audio or video calls (WebRTC)?
V17.2.1	Does the app have real-time audio or video calls (WebRTC)?
V17.2.2	Does the app have real-time audio or video calls (WebRTC)?
V17.2.3	Does the app have real-time audio or video calls (WebRTC)?
V17.2.4	Does the app have real-time audio or video calls (WebRTC)?
V17.3.1	Does the app have real-time audio or video calls (WebRTC)?
V17.3.2	Does the app have real-time audio or video calls (WebRTC)?
V3.5.4	Do other applications share this app's hostname?
V6.6.1	Does the app send sign-in codes by phone, SMS, or push notification?

requirement	what has to be answered
V6.6.2	Does the app send sign-in codes by phone, SMS, push notification, or email?
V6.8.1	Does the app sign people in through another provider (OAuth or OpenID Connect)?
V6.8.2	Does the app sign people in through another provider (OAuth or OpenID Connect)?
V6.8.4	Does the app sign people in through another provider (OAuth or OpenID Connect)?
V7.1.3	Does the app sign people in through another provider (OAuth or OpenID Connect)?
V7.6.1	Does the app sign people in through another provider (OAuth or OpenID Connect)?
V7.6.2	Does the app sign people in through another provider (OAuth or OpenID Connect)?
V8.4.1	Does the app serve several separate customer organizations that share one system?
V9.1.1	Does the app issue self-contained tokens such as JWTs?
V9.1.2	Does the app issue self-contained tokens such as JWTs?
V9.1.3	Does the app issue self-contained tokens such as JWTs?
V9.2.1	Does the app issue self-contained tokens such as JWTs?
V9.2.2	Does the app issue self-contained tokens such as JWTs?
V9.2.3	Does the app issue self-contained tokens such as JWTs?
V9.2.4	Does the app issue self-contained tokens such as JWTs?

Checks that ran and found nothing, against nothing in the tables above

These were satisfied. They appear here rather than beside a requirement because what they look at is not something this app is being assessed on.

check	what it covered	why it is here
config.mcp-server-unpinned	3 files: no MCP server started with npx, uvx, or docker from a package or image named in them	it names no requirement in any loaded framework
config.certificate-checks-off	3 files: none sets `NODE_TLS_REJECT_UNAUTHORIZED` or `PYTHONHTTPSVERIFY` to `0` or switches certificate checking off for the whole app in code; the same setting made on the server itself is in no file	it names no requirement in any loaded framework
config.rich-text-without-sanitizer	0 packages: none is a rich-text editor `sv` knows	it names no requirement in any loaded framework
config.retired-grant-enabled	no sign-in server library `sv` knows (django-oauth-toolkit, Doorkeeper, fosite, node-oauth2-server, league/oauth2-server, Laravel Passport) is used in the app's files	it names no requirement in any loaded framework
config.model-file-can-run-code	0 file(s) named like a model file, none holding a pickle; 2 other file(s) that are not code, none opening as a pickle; a model downloaded when the app runs is not seen	it names no requirement in any loaded framework

Requirements that do not apply, and why — reference: why each was ruled out

An exclusion resting on the manifest's word is weaker than one resting on what the code actually contains. The last column says which this is.

requirement	why not	rests on
V1.2.6	No LDAP client library is used, so LDAP injection is not possible.	what the app's code contains (ldap)
V1.2.7	No XPath library is used.	what the app's code contains (xpath)
V1.2.8	No LaTeX processing library is used.	what the app's code contains (latex)
V1.3.8	JNDI is a Java technology and this app does not run on the JVM.	what the app's code contains (jndi)
V1.3.9	No memcache client library is used.	what the app's code contains (memcache)
V1.4.1	This app's languages are memory-safe and it contains no unmanaged (C/C++) code.	what the app's code contains (unmanaged-code)
V1.4.2	This app's languages are memory-safe and it contains no unmanaged (C/C++) code.	what the app's code contains (unmanaged-code)
V1.4.3	This app's languages are memory-safe and it contains no unmanaged (C/C++) code.	what the app's code contains (unmanaged-code)
V1.5.1	No XML parser is used, so XML external entity (XXE) attacks are not possible.	what the app's code contains (xml)
V12.1.3	The app asks for no client certificate, so there is no certificate identity to validate before it is used.	what the app's code contains (mtls)
V3.5.5	This app's pages do not use the browser postMessage feature.	what the app's code contains (postmessage)
V4.3.1	No GraphQL library is used.	what the app's code contains (graphql)
V4.3.2	No GraphQL library is used.	what the app's code contains (graphql)
V4.4.1	No WebSocket library is used; every request is a normal web request.	what the app's code contains (websockets)
V4.4.2	No WebSocket library is used; every request is a normal web request.	what the app's code contains (websockets)
V4.4.3	No WebSocket library is used; every request is a normal web request.	what the app's code contains (websockets)
V4.4.4	No WebSocket library is used; every request is a normal web request.	what the app's code contains (websockets)
V6.8.3	No SAML library is used, so there are no SAML assertions to replay.	what the app's code contains (saml)

What the app says about itself

about	stackvet.toml	who said it	the code	what that means
auth	yes	not recorded, so read as your AI coding tool's	nothing	Taken on the manifest's word. `sv` looked and found nothing, which for this is not the same as finding it absent.
oauth	nothing	nobody	nothing	Nobody has said. Requirements that turn on this are not assessed rather than excluded.
authorization-server	nothing	nobody	nothing	Nobody has said. Requirements that turn on this are not assessed rather than excluded.
mcp-server	nothing	nobody	nothing	Nobody has said. Requirements that turn on this are not assessed rather than excluded.

about	stackvet.toml	who said it	the code	what that means
jwt	nothing	nobody	nothing	Nobody has said. Requirements that turn on this are not assessed rather than excluded.
uploads	yes	not recorded, so read as your AI coding tool's	nothing	Taken on the manifest's word. `sv` looked and found nothing, which for this is not the same as finding it absent.
payments	nothing	nobody	nothing	Nobody has said. Requirements that turn on this are not assessed rather than excluded.
email	nothing	nobody	nothing	Nobody has said. Requirements that turn on this are not assessed rather than excluded.
public-api	nothing	nobody	nothing	Nobody has said. Requirements that turn on this are not assessed rather than excluded.
scheduler	nothing	nobody	nothing	Nobody has said. Requirements that turn on this are not assessed rather than excluded.
multi-tenant	nothing	nobody	nothing	Nobody has said. Requirements that turn on this are not assessed rather than excluded.
multiple-services	nothing	nobody	nothing	Nobody has said. Requirements that turn on this are not assessed rather than excluded.
webrtc	nothing	nobody	nothing	Nobody has said. Requirements that turn on this are not assessed rather than excluded.
external-apis	nothing	nobody	nothing	Nobody has said. Requirements that turn on this are not assessed rather than excluded.
tls	yes	not recorded, so read as your AI coding tool's	nothing	Taken on the manifest's word. `sv` looked and found nothing, which for this is not the same as finding it absent.
internet	yes	not recorded, so read as your AI coding tool's	yes	The manifest and the code agree.
ai	nothing	nobody	nothing	Nobody has said. Requirements that turn on this are not assessed rather than excluded.
ai-actions	nothing	nobody	nothing	Nobody has said. Requirements that turn on this are not assessed rather than excluded.
ai-history	nothing	nobody	nothing	Nobody has said. Requirements that turn on this are not assessed rather than excluded.
ai-moderation	nothing	nobody	nothing	Nobody has said. Requirements that turn on this are not assessed rather than excluded.
rag	nothing	nobody	nothing	Nobody has said. Requirements that turn on this are not assessed rather than excluded.
web-search	nothing	nobody	nothing	Nobody has said. Requirements that turn on this are not assessed rather than excluded.
generates-media	nothing	nobody	nothing	Nobody has said. Requirements that turn on this are not assessed rather than excluded.
mcp	nothing	nobody	nothing	Nobody has said. Requirements that turn on this are not assessed rather than excluded.
training	nothing	nobody	nothing	Nobody has said. Requirements that turn on this are not assessed rather than excluded.

about	stackvet.toml	who said it	the code	what that means
level2	yes	not recorded, so read as your AI coding tool's	nothing	Taken on the manifest's word. `sv` looked and found nothing, which for this is not the same as finding it absent.
self-hosted-model	nothing	nobody	nothing	Nobody has said. Requirements that turn on this are not assessed rather than excluded.
multi-agent	nothing	nobody	nothing	Nobody has said. Requirements that turn on this are not assessed rather than excluded.
multimodal-ai	nothing	nobody	nothing	Nobody has said. Requirements that turn on this are not assessed rather than excluded.
out-of-band-auth	nothing	nobody	nothing	Nobody has said. Requirements that turn on this are not assessed rather than excluded.
out-of-band-or-email-code	nothing	nobody	nothing	Nobody has said. Requirements that turn on this are not assessed rather than excluded.
shared-hostname	nothing	nobody	nothing	Nobody has said. Requirements that turn on this are not assessed rather than excluded.
rich-text	nothing	nobody	nothing	Nobody has said. Requirements that turn on this are not assessed rather than excluded.
ci-cd	nothing	nobody	no	Nobody has said. Requirements that turn on this are not assessed rather than excluded.
hosted-scm	nothing	nobody	nothing	Nobody has said. Requirements that turn on this are not assessed rather than excluded.
outside-contributors	nothing	nobody	nothing	Nobody has said. Requirements that turn on this are not assessed rather than excluded.
iac	nothing	nobody	no	Nobody has said. Requirements that turn on this are not assessed rather than excluded.
websockets	nothing	nobody	no	The manifest and the code agree.
graphql	nothing	nobody	no	The manifest and the code agree.
ldap	nothing	nobody	no	The manifest and the code agree.
xpath	nothing	nobody	no	The manifest and the code agree.
xml	nothing	nobody	no	The manifest and the code agree.
mtls	nothing	nobody	no	The manifest and the code agree.
latex	nothing	nobody	no	The manifest and the code agree.
jndi	nothing	nobody	no	The manifest and the code agree.
memcache	nothing	nobody	no	The manifest and the code agree.
format-strings	nothing	nobody	yes	The manifest and the code agree.
unmanaged-code	nothing	nobody	no	The manifest and the code agree.
postmessage	nothing	nobody	no	The manifest and the code agree.
saml	nothing	nobody	no	The manifest and the code agree.
cryptographic-authenticator	nothing	nobody	no	The manifest and the code agree.

Every low-severity finding

1 finding, grouped by rule above. Each counts as a finding.

[low] There is no way to report a security problem

Where: `SECURITY.md` line 1

How sure: confirmed.

Fingerprint: `882dedc677bff5a5`*. A person who has looked and found it a false alarm, or a risk to live with for now, can set it aside under* `[[finding-review]]` *in stackvet.toml.*

Next step. Fix it in the code and check again. If it is a false alarm, write a `[[finding-review]]` entry for it, naming its rule, file and fingerprint, and confirm it with `sv review` in your own terminal; until then it counts.

No SECURITY.md or security.txt was found, so somebody who finds a problem in this app has nowhere obvious to say so.

Why it matters. Problems found by outsiders get reported publicly, or not at all.

What to do. Add a SECURITY.md saying where to send a report and how long a reply should take.

Appendix: what each requirement asks for

The words of the standard, for every requirement that applies to this app, in the order of the chapters above.

requirement	what it asks for
V1.2.1	Verify that output encoding for an HTTP response, HTML document, or XML document is relevant for the context required, such as encoding the relevant characters for HTML elements, HTML attributes, HTML comments, CSS, or HTTP header fields, to avoid changing the message or document structure.
V1.2.2	Verify that when dynamically building URLs, untrusted data is encoded according to its context (e.g., URL encoding or base64url encoding for query or path parameters). Ensure that only safe URL protocols are permitted (e.g., disallow javascript: or data:).
V1.2.3	Verify that output encoding or escaping is used when dynamically building JavaScript content (including JSON), to avoid changing the message or document structure (to avoid JavaScript and JSON injection).
V1.2.4	Verify that data selection or database queries (e.g., SQL, HQL, NoSQL, Cypher) use parameterized queries, ORMs, entity frameworks, or are otherwise protected from SQL Injection and other database injection attacks. This is also relevant when writing stored procedures.
V1.2.5	Verify that the application protects against OS command injection and that operating system calls use parameterized OS queries or use contextual command line output encoding.
V1.3.2	Verify that the application avoids the use of eval() or other dynamic code execution features such as Spring Expression Language (SpEL). Where there is no alternative, any user input being included must be sanitized before being executed.
V1.1.1	Verify that input is decoded or unescaped into a canonical form only once, it is only decoded when encoded data in that form is expected, and that this is done before processing the input further, for example it is not performed after input validation or sanitization.
V1.1.2	Verify that the application performs output encoding and escaping either as a final step before being used by the interpreter for which it is intended or by the interpreter itself.
V1.2.9	Verify that the application escapes special characters in regular expressions (typically using a backslash) to prevent them from being misinterpreted as metacharacters.

requirement	what it asks for
V1.3.3	Verify that data being passed to a potentially dangerous context is sanitized beforehand to enforce safety measures, such as only allowing characters which are safe for this context and trimming input which is too long.
V1.3.4	Verify that user-supplied Scalable Vector Graphics (SVG) scriptable content is validated or sanitized to contain only tags and attributes (such as draw graphics) that are safe for the application, e.g., do not contain scripts and foreignObject.
V1.3.6	Verify that the application protects against Server-side Request Forgery (SSRF) attacks, by validating untrusted data against an allowlist of protocols, domains, paths and ports and sanitizing potentially dangerous characters before using the data to call another service.
V1.3.7	Verify that the application protects against template injection attacks by not allowing templates to be built based on untrusted input. Where there is no alternative, any untrusted input being included dynamically during template creation must be sanitized or strictly validated.
V1.3.10	Verify that format strings which might resolve in an unexpected or malicious way when used are sanitized before being processed.
V1.5.2	Verify that deserialization of untrusted data enforces safe input handling, such as using an allowlist of object types or restricting client-defined object types, to prevent deserialization attacks. Deserialization mechanisms that are explicitly defined as insecure must not be used with untrusted input.
V2.1.1	Verify that the application's documentation defines input validation rules for how to check the validity of data items against an expected structure. This could be common data formats such as credit card numbers, email addresses, telephone numbers, or it could be an internal data format.
V2.2.1	Verify that input is validated to enforce business or functional expectations for that input. This should either use positive validation against an allow list of values, patterns, and ranges, or be based on comparing the input to an expected structure and logical limits according to predefined rules. For L1, this can focus on input which is used to make specific business or security decisions. For L2 and up, this should apply to all input.
V2.2.2	Verify that the application is designed to enforce input validation at a trusted service layer. While client-side validation improves usability and should be encouraged, it must not be relied upon as a security control.
V2.3.1	Verify that the application will only process business logic flows for the same user in the expected sequential step order and without skipping steps.
V2.1.2	Verify that the application's documentation defines how to validate the logical and contextual consistency of combined data items, such as checking that suburb and ZIP code match.
V2.1.3	Verify that expectations for business logic limits and validations are documented, including both per-user and globally across the application.
V2.2.3	Verify that the application ensures that combinations of related data items are reasonable according to the pre-defined rules.
V2.3.2	Verify that business logic limits are implemented per the application's documentation to avoid business logic flaws being exploited.
V2.3.3	Verify that transactions are being used at the business logic level such that either a business logic operation succeeds in its entirety or it is rolled back to the previous correct state.
V2.3.4	Verify that business logic level locking mechanisms are used to ensure that limited quantity resources (such as theater seats or delivery slots) cannot be double-booked by manipulating the application's logic.
V2.4.1	Verify that anti-automation controls are in place to protect against excessive calls to application functions that could lead to data exfiltration, garbage-data creation, quota exhaustion, rate-limit breaches, denial-of-service, or overuse of costly resources.

requirement	what it asks for
V3.2.1	Verify that security controls are in place to prevent browsers from rendering content or functionality in HTTP responses in an incorrect context (e.g., when an API, a user-uploaded file or other resource is requested directly). Possible controls could include: not serving the content unless HTTP request header fields (such as Sec-Fetch-\) *indicate it is the correct context, using the sandbox directive of the Content-Security-Policy header field or using the attachment disposition type in the Content-Disposition header field.*
V3.2.2	Verify that content intended to be displayed as text, rather than rendered as HTML, is handled using safe rendering functions (such as createTextNode or textContent) to prevent unintended execution of content such as HTML or JavaScript.
V3.3.1	Verify that cookies have the 'Secure' attribute set, and if the '__Host-' prefix is not used for the cookie name, the '__Secure-' prefix must be used for the cookie name.
V3.4.1	Verify that a Strict-Transport-Security header field is included on all responses to enforce an HTTP Strict Transport Security (HSTS) policy. A maximum age of at least 1 year must be defined, and for L2 and up, the policy must apply to all subdomains as well.
V3.4.2	Verify that the Cross-Origin Resource Sharing (CORS) Access-Control-Allow-Origin header field is a fixed value by the application, or if the Origin HTTP request header field value is used, it is validated against an allowlist of trusted origins. When 'Access-Control-Allow-Origin: *' needs to be used, verify that the response does not include any sensitive information.*
V3.5.1	Verify that, if the application does not rely on the CORS preflight mechanism to prevent disallowed cross-origin requests to use sensitive functionality, these requests are validated to ensure they originate from the application itself. This may be done by using and validating anti-forgery tokens or requiring extra HTTP header fields that are not CORS-safelisted request-header fields. This is to defend against browser-based request forgery attacks, commonly known as cross-site request forgery (CSRF).
V3.5.2	Verify that, if the application relies on the CORS preflight mechanism to prevent disallowed cross-origin use of sensitive functionality, it is not possible to call the functionality with a request which does not trigger a CORS-preflight request. This may require checking the values of the 'Origin' and 'Content-Type' request header fields or using an extra header field that is not a CORS-safelisted header-field.
V3.5.3	Verify that HTTP requests to sensitive functionality use appropriate HTTP methods such as POST, PUT, PATCH, or DELETE, and not methods defined by the HTTP specification as "safe" such as HEAD, OPTIONS, or GET. Alternatively, strict validation of the Sec-Fetch-* request header fields can be used to ensure that the request did not originate from an inappropriate cross-origin call, a navigation request, or a resource load (such as an image source) where this is not expected.
V3.3.2	Verify that each cookie's 'SameSite' attribute value is set according to the purpose of the cookie, to limit exposure to user interface redress attacks and browser-based request forgery attacks, commonly known as cross-site request forgery (CSRF).
V3.3.3	Verify that cookies have the '__Host-' prefix for the cookie name unless they are explicitly designed to be shared with other hosts.
V3.3.4	Verify that if the value of a cookie is not meant to be accessible to client-side scripts (such as a session token), the cookie must have the 'HttpOnly' attribute set and the same value (e. g. session token) must only be transferred to the client via the 'Set-Cookie' header field.
V3.4.3	Verify that HTTP responses include a Content-Security-Policy response header field which defines directives to ensure the browser only loads and executes trusted content or resources, in order to limit execution of malicious JavaScript. As a minimum, a global policy must be used which includes the directives object-src 'none' and base-uri 'none' and defines either an allowlist or uses nonces or hashes. For an L3 application, a per-response policy with nonces or hashes must be defined.
V3.4.4	Verify that all HTTP responses contain an 'X-Content-Type-Options: nosniff' header field. This instructs browsers not to use content sniffing and MIME type guessing for the given response, and to require the response's Content-Type header field value to match the destination resource. For example, the response to a request for a style is only accepted if the response's Content-Type is 'text/css'. This also enables the use of the Cross-Origin Read Blocking (CORB) functionality by the browser.

requirement	what it asks for
V3.4.5	Verify that the application sets a referrer policy to prevent leakage of technically sensitive data to third-party services via the 'Referer' HTTP request header field. This can be done using the Referrer-Policy HTTP response header field or via HTML element attributes. Sensitive data could include path and query data in the URL, and for internal non-public applications also the hostname.
V3.4.6	Verify that the web application uses the frame-ancestors directive of the Content-Security-Policy header field for every HTTP response to ensure that it cannot be embedded by default and that embedding of specific resources is allowed only when necessary. Note that the X-Frame-Options header field, although supported by browsers, is obsolete and may not be relied upon.
V3.7.1	Verify that the application only uses client-side technologies which are still supported and considered secure. Examples of technologies which do not meet this requirement include NSAPI plugins, Flash, Shockwave, ActiveX, Silverlight, NACL, or client-side Java applets.
V3.7.2	Verify that the application will only automatically redirect the user to a different hostname or domain (which is not controlled by the application) where the destination appears on an allowlist.
V4.1.1	Verify that every HTTP response with a message body contains a Content-Type header field that matches the actual content of the response, including the charset parameter to specify safe character encoding (e.g., UTF-8, ISO-8859-1) according to IANA Media Types, such as "text/", "/+xml" and "/xml".
V4.1.2	Verify that only user-facing endpoints (intended for manual web-browser access) automatically redirect from HTTP to HTTPS, while other services or endpoints do not implement transparent redirects. This is to avoid a situation where a client is erroneously sending unencrypted HTTP requests, but since the requests are being automatically redirected to HTTPS, the leakage of sensitive data goes undiscovered.
V4.1.3	Verify that any HTTP header field used by the application and set by an intermediary layer, such as a load balancer, a web proxy, or a backend-for-frontend service, cannot be overridden by the end-user. Example headers might include X-Real-IP, X-Forwarded-*, or X-User-ID.*
V4.2.1	Verify that all application components (including load balancers, firewalls, and application servers) determine boundaries of incoming HTTP messages using the appropriate mechanism for the HTTP version to prevent HTTP request smuggling. In HTTP/1.x, if a Transfer-Encoding header field is present, the Content-Length header must be ignored per RFC 2616. When using HTTP/2 or HTTP/3, if a Content-Length header field is present, the receiver must ensure that it is consistent with the length of the DATA frames.
V5.2.1	Verify that the application will only accept files of a size which it can process without causing a loss of performance or a denial of service attack.
V5.2.2	Verify that when the application accepts a file, either on its own or within an archive such as a zip file, it checks if the file extension matches an expected file extension and validates that the contents correspond to the type represented by the extension. This includes, but is not limited to, checking the initial 'magic bytes', performing image re-writing, and using specialized libraries for file content validation. For L1, this can focus just on files which are used to make specific business or security decisions. For L2 and up, this must apply to all files being accepted.
V5.3.1	Verify that files uploaded or generated by untrusted input and stored in a public folder, are not executed as server-side program code when accessed directly with an HTTP request.
V5.3.2	Verify that when the application creates file paths for file operations, instead of user-submitted filenames, it uses internally generated or trusted data, or if user-submitted filenames or file metadata must be used, strict validation and sanitization must be applied. This is to protect against path traversal, local or remote file inclusion (LFI, RFI), and server-side request forgery (SSRF) attacks.
V5.1.1	Verify that the documentation defines the permitted file types, expected file extensions, and maximum size (including unpacked size) for each upload feature. Additionally, ensure that the documentation specifies how files are made safe for end-users to download and process, such as how the application behaves when a malicious file is detected.
V5.2.3	Verify that the application checks compressed files (e.g., zip, gz, docx, odt) against maximum allowed uncompressed size and against maximum number of files before uncompressing the file.
V5.4.1	Verify that the application validates or ignores user-submitted filenames, including in a JSON, JSONP, or URL parameter and specifies a filename in the Content-Disposition header field in the response.

requirement	what it asks for
V5.4.2	Verify that file names served (e.g., in HTTP response header fields or email attachments) are encoded or sanitized (e.g., following RFC 6266) to preserve document structure and prevent injection attacks.
V5.4.3	Verify that files obtained from untrusted sources are scanned by antivirus scanners to prevent serving of known malicious content.
V6.1.1	Verify that application documentation defines how controls such as rate limiting, anti-automation, and adaptive response, are used to defend against attacks such as credential stuffing and password brute force. The documentation must make clear how these controls are configured and prevent malicious account lockout.
V6.2.1	Verify that user set passwords are at least 8 characters in length although a minimum of 15 characters is strongly recommended.
V6.2.2	Verify that users can change their password.
V6.2.3	Verify that password change functionality requires the user's current and new password.
V6.2.4	Verify that passwords submitted during account registration or password change are checked against an available set of, at least, the top 3000 passwords which match the application's password policy, e.g. minimum length.
V6.2.5	Verify that passwords of any composition can be used, without rules limiting the type of characters permitted. There must be no requirement for a minimum number of upper or lower case characters, numbers, or special characters.
V6.2.6	Verify that password input fields use type=password to mask the entry. Applications may allow the user to temporarily view the entire masked password, or the last typed character of the password.
V6.2.7	Verify that "paste" functionality, browser password helpers, and external password managers are permitted.
V6.2.8	Verify that the application verifies the user's password exactly as received from the user, without any modifications such as truncation or case transformation.
V6.3.1	Verify that controls to prevent attacks such as credential stuffing and password brute force are implemented according to the application's security documentation.
V6.3.2	Verify that default user accounts (e.g., "root", "admin", or "sa") are not present in the application or are disabled.
V6.4.1	Verify that system generated initial passwords or activation codes are securely randomly generated, follow the existing password policy, and expire after a short period of time or after they are initially used. These initial secrets must not be permitted to become the long term password.
V6.4.2	Verify that password hints or knowledge-based authentication (so-called "secret questions") are not present.
V6.1.2	Verify that a list of context-specific words is documented in order to prevent their use in passwords. The list could include permutations of organization names, product names, system identifiers, project codenames, department or role names, and similar.
V6.1.3	Verify that, if the application includes multiple authentication pathways, these are all documented together with the security controls and authentication strength which must be consistently enforced across them.
V6.2.9	Verify that passwords of at least 64 characters are permitted.
V6.2.10	Verify that a user's password stays valid until it is discovered to be compromised or the user rotates it. The application must not require periodic credential rotation.
V6.2.11	Verify that the documented list of context specific words is used to prevent easy to guess passwords being created.
V6.2.12	Verify that passwords submitted during account registration or password changes are checked against a set of breached passwords.
V6.3.3	Verify that either a multi-factor authentication mechanism or a combination of single-factor authentication mechanisms, must be used in order to access the application. For L3, one of the factors must be a hardware-based authentication mechanism which provides compromise and impersonation resistance against phishing attacks while verifying the intent to authenticate by requiring a user-initiated action (such as a button press on a FIDO hardware key or a mobile phone). Relaxing any of the considerations in this requirement requires a fully documented rationale and a comprehensive set of mitigating controls.

requirement	what it asks for
V6.3.4	Verify that, if the application includes multiple authentication pathways, there are no undocumented pathways and that security controls and authentication strength are enforced consistently.
V6.4.3	Verify that a secure process for resetting a forgotten password is implemented, that does not bypass any enabled multi-factor authentication mechanisms.
V6.4.4	Verify that if a multi-factor authentication factor is lost, evidence of identity proofing is performed at the same level as during enrollment.
V6.5.1	Verify that lookup secrets, out-of-band authentication requests or codes, and time-based one-time passwords (TOTPs) are only successfully usable once.
V6.5.2	Verify that, when being stored in the application's backend, lookup secrets with less than 112 bits of entropy (19 random alphanumeric characters or 34 random digits) are hashed with an approved password storage hashing algorithm that incorporates a 32-bit random salt. A standard hash function can be used if the secret has 112 bits of entropy or more.
V6.5.3	Verify that lookup secrets, out-of-band authentication code, and time-based one-time password seeds, are generated using a Cryptographically Secure Pseudorandom Number Generator (CSPRNG) to avoid predictable values.
V6.5.4	Verify that lookup secrets and out-of-band authentication codes have a minimum of 20 bits of entropy (typically 4 random alphanumeric characters or 6 random digits is sufficient).
V6.5.5	Verify that out-of-band authentication requests, codes, or tokens, as well as time-based one-time passwords (TOTPs) have a defined lifetime. Out of band requests must have a maximum lifetime of 10 minutes and for TOTP a maximum lifetime of 30 seconds.
V6.6.3	Verify that a code based out-of-band authentication mechanism is protected against brute force attacks by using rate limiting. Consider also using a code with at least 64 bits of entropy.
V7.2.1	Verify that the application performs all session token verification using a trusted, backend service.
V7.2.2	Verify that the application uses either self-contained or reference tokens that are dynamically generated for session management, i.e. not using static API secrets and keys.
V7.2.3	Verify that if reference tokens are used to represent user sessions, they are unique and generated using a cryptographically secure pseudo-random number generator (CSPRNG) and possess at least 128 bits of entropy.
V7.2.4	Verify that the application generates a new session token on user authentication, including re-authentication, and terminates the current session token.
V7.4.1	Verify that when session termination is triggered (such as logout or expiration), the application disallows any further use of the session. For reference tokens or stateful sessions, this means invalidating the session data at the application backend. Applications using self-contained tokens will need a solution such as maintaining a list of terminated tokens, disallowing tokens produced before a per-user date and time or rotating a per-user signing key.
V7.4.2	Verify that the application terminates all active sessions when a user account is disabled or deleted (such as an employee leaving the company).
V7.1.1	Verify that the user's session inactivity timeout and absolute maximum session lifetime are documented, are appropriate in combination with other controls, and that the documentation includes justification for any deviations from NIST SP 800-63B re-authentication requirements.
V7.1.2	Verify that the documentation defines how many concurrent (parallel) sessions are allowed for one account as well as the intended behaviors and actions to be taken when the maximum number of active sessions is reached.
V7.3.1	Verify that there is an inactivity timeout such that re-authentication is enforced according to risk analysis and documented security decisions.
V7.3.2	Verify that there is an absolute maximum session lifetime such that re-authentication is enforced according to risk analysis and documented security decisions.

requirement	what it asks for
V7.4.3	Verify that the application gives the option to terminate all other active sessions after a successful change or removal of any authentication factor (including password change via reset or recovery and, if present, an MFA settings update).
V7.4.4	Verify that all pages that require authentication have easy and visible access to logout functionality.
V7.4.5	Verify that application administrators are able to terminate active sessions for an individual user or for all users.
V7.5.1	Verify that the application requires full re-authentication before allowing modifications to sensitive account attributes which may affect authentication such as email address, phone number, MFA configuration, or other information used in account recovery.
V7.5.2	Verify that users are able to view and (having authenticated again with at least one factor) terminate any or all currently active sessions.
V8.1.1	Verify that authorization documentation defines rules for restricting function-level and data-specific access based on consumer permissions and resource attributes.
V8.2.1	Verify that the application ensures that function-level access is restricted to consumers with explicit permissions.
V8.2.2	Verify that the application ensures that data-specific access is restricted to consumers with explicit permissions to specific data items to mitigate insecure direct object reference (IDOR) and broken object level authorization (BOLA).
V8.3.1	Verify that the application enforces authorization rules at a trusted service layer and doesn't rely on controls that an untrusted consumer could manipulate, such as client-side JavaScript.
V8.1.2	Verify that authorization documentation defines rules for field-level access restrictions (both read and write) based on consumer permissions and resource attributes. Note that these rules might depend on other attribute values of the relevant data object, such as state or status.
V8.2.3	Verify that the application ensures that field-level access is restricted to consumers with explicit permissions to specific fields to mitigate broken object property level authorization (BOPLA).
V11.3.1	Verify that insecure block modes (e.g., ECB) and weak padding schemes (e.g., PKCS#1 v1.5) are not used.
V11.3.2	Verify that only approved ciphers and modes such as AES with GCM are used.
V11.4.1	Verify that only approved hash functions are used for general cryptographic use cases, including digital signatures, HMAC, KDF, and random bit generation. Disallowed hash functions, such as MD5, must not be used for any cryptographic purpose.
V11.1.1	Verify that there is a documented policy for management of cryptographic keys and a cryptographic key lifecycle that follows a key management standard such as NIST SP 800-57. This should include ensuring that keys are not overshared (for example, with more than two entities for shared secrets and more than one entity for private keys).
V11.1.2	Verify that a cryptographic inventory is performed, maintained, regularly updated, and includes all cryptographic keys, algorithms, and certificates used by the application. It must also document where keys can and cannot be used in the system, and the types of data that can and cannot be protected using the keys.
V11.2.1	Verify that industry-validated implementations (including libraries and hardware-accelerated implementations) are used for cryptographic operations.
V11.2.2	Verify that the application is designed with crypto agility such that random number, authenticated encryption, MAC, or hashing algorithms, key lengths, rounds, ciphers and modes can be reconfigured, upgraded, or swapped at any time, to protect against cryptographic breaks. Similarly, it must also be possible to replace keys and passwords and re-encrypt data. This will allow for seamless upgrades to post-quantum cryptography (PQC), once high-assurance implementations of approved PQC schemes or standards are widely available.
V11.2.3	Verify that all cryptographic primitives utilize a minimum of 128-bits of security based on the algorithm, key size, and configuration. For example, a 256-bit ECC key provides roughly 128 bits of security where RSA requires a 3072-bit key to achieve 128 bits of security.
V11.3.3	Verify that encrypted data is protected against unauthorized modification preferably by using an approved authenticated encryption method or by combining an approved encryption method with an approved MAC algorithm.

requirement	what it asks for
V11.4.2	Verify that passwords are stored using an approved, computationally intensive, key derivation function (also known as a "password hashing function"), with parameter settings configured based on current guidance. The settings should balance security and performance to make brute-force attacks sufficiently challenging for the required level of security.
V11.4.3	Verify that hash functions used in digital signatures, as part of data authentication or data integrity are collision resistant and have appropriate bit-lengths. If collision resistance is required, the output length must be at least 256 bits. If only resistance to second pre-image attacks is required, the output length must be at least 128 bits.
V11.4.4	Verify that the application uses approved key derivation functions with key stretching parameters when deriving secret keys from passwords. The parameters in use must balance security and performance to prevent brute-force attacks from compromising the resulting cryptographic key.
V11.5.1	Verify that all random numbers and strings which are intended to be non-guessable must be generated using a cryptographically secure pseudo-random number generator (CSPRNG) and have at least 128 bits of entropy. Note that UUIDs do not respect this condition.
V11.6.1	Verify that only approved cryptographic algorithms and modes of operation are used for key generation and seeding, and digital signature generation and verification. Key generation algorithms must not generate insecure keys vulnerable to known attacks, for example, RSA keys which are vulnerable to Fermat factorization.
V12.1.1	Verify that only the latest recommended versions of the TLS protocol are enabled, such as TLS 1.2 and TLS 1.3. The latest version of the TLS protocol must be the preferred option.
V12.2.1	Verify that TLS is used for all connectivity between a client and external facing, HTTP-based services, and does not fall back to insecure or unencrypted communications.
V12.2.2	Verify that external facing services use publicly trusted TLS certificates.
V12.1.2	Verify that only recommended cipher suites are enabled, with the strongest cipher suites set as preferred. L3 applications must only support cipher suites which provide forward secrecy.
V13.4.1	Verify that the application is deployed either without any source control metadata, including the .git or .svn folders, or in a way that these folders are inaccessible both externally and to the application itself.
V13.1.1	Verify that all communication needs for the application are documented. This must include external services which the application relies upon and cases where an end user might be able to provide an external location to which the application will then connect.
V13.2.3	Verify that if a credential has to be used for service authentication, the credential being used by the consumer is not a default credential (e.g., root/root or admin/admin).
V13.3.1	Verify that a secrets management solution, such as a key vault, is used to securely create, store, control access to, and destroy backend secrets. These could include passwords, key material, integrations with databases and third-party systems, keys and seeds for time-based tokens, other internal secrets, and API keys. Secrets must not be included in application source code or included in build artifacts. For an L3 application, this must involve a hardware-backed solution such as an HSM.
V13.3.2	Verify that access to secret assets adheres to the principle of least privilege.
V13.4.2	Verify that debug modes are disabled for all components in production environments to prevent exposure of debugging features and information leakage.
V13.4.3	Verify that web servers do not expose directory listings to clients unless explicitly intended.
V13.4.4	Verify that using the HTTP TRACE method is not supported in production environments, to avoid potential information leakage.
V13.4.5	Verify that documentation (such as for internal APIs) and monitoring endpoints are not exposed unless explicitly intended.
V14.2.1	Verify that sensitive data is only sent to the server in the HTTP message body or header fields, and that the URL and query string do not contain sensitive information, such as an API key or session token.

requirement	what it asks for
V14.3.1	Verify that authenticated data is cleared from client storage, such as the browser DOM, after the client or session is terminated. The 'Clear-Site-Data' HTTP response header field may be able to help with this but the client-side should also be able to clear up if the server connection is not available when the session is terminated.
V14.1.1	Verify that all sensitive data created and processed by the application has been identified and classified into protection levels. This includes data that is only encoded and therefore easily decoded, such as Base64 strings or the plaintext payload inside a JWT. Protection levels need to take into account any data protection and privacy regulations and standards which the application is required to comply with.
V14.1.2	Verify that all sensitive data protection levels have a documented set of protection requirements. This must include (but not be limited to) requirements related to general encryption, integrity verification, retention, how the data is to be logged, access controls around sensitive data in logs, database-level encryption, privacy and privacy-enhancing technologies to be used, and other confidentiality requirements.
V14.2.2	Verify that the application prevents sensitive data from being cached in server components, such as load balancers and application caches, or ensures that the data is securely purged after use.
V14.2.3	Verify that defined sensitive data is not sent to untrusted parties (e.g., user trackers) to prevent unwanted collection of data outside of the application's control.
V14.2.4	Verify that controls around sensitive data related to encryption, integrity verification, retention, how the data is to be logged, access controls around sensitive data in logs, privacy and privacy-enhancing technologies, are implemented as defined in the documentation for the specific data's protection level.
V14.3.2	Verify that the application sets sufficient anti-caching HTTP response header fields (i.e., Cache-Control: no-store) so that sensitive data is not cached in browsers.
V14.3.3	Verify that data stored in browser storage (such as localStorage, sessionStorage, IndexedDB, or cookies) does not contain sensitive data, with the exception of session tokens.
V15.1.1	Verify that application documentation defines risk based remediation time frames for 3rd party component versions with vulnerabilities and for updating libraries in general, to minimize the risk from these components.
V15.2.1	Verify that the application only contains components which have not breached the documented update and remediation time frames.
V15.3.1	Verify that the application only returns the required subset of fields from a data object. For example, it should not return an entire data object, as some individual fields should not be accessible to users.
V15.1.2	Verify that an inventory catalog, such as software bill of materials (SBOM), is maintained of all third-party libraries in use, including verifying that components come from pre-defined, trusted, and continually maintained repositories.
V15.1.3	Verify that the application documentation identifies functionality which is time-consuming or resource-demanding. This must include how to prevent a loss of availability due to overusing this functionality and how to avoid a situation where building a response takes longer than the consumer's timeout. Potential defenses may include asynchronous processing, using queues, and limiting parallel processes per user and per application.
V15.2.2	Verify that the application has implemented defenses against loss of availability due to functionality which is time-consuming or resource-demanding, based on the documented security decisions and strategies for this.
V15.2.3	Verify that the production environment only includes functionality that is required for the application to function, and does not expose extraneous functionality such as test code, sample snippets, and development functionality.
V15.3.3	Verify that the application has countermeasures to protect against mass assignment attacks by limiting allowed fields per controller and action, e.g., it is not possible to insert or update a field value when it was not intended to be part of that action.
V15.3.4	Verify that all proxying and middleware components transfer the user's original IP address correctly using trusted data fields that cannot be manipulated by the end user, and the application and web server use this correct value for logging and security decisions such as rate limiting, taking into account that even the original IP address may not be reliable due to dynamic IPs, VPNs, or corporate firewalls.

requirement	what it asks for
V15.3.5	Verify that the application explicitly ensures that variables are of the correct type and performs strict equality and comparator operations. This is to avoid type juggling or type confusion vulnerabilities caused by the application code making an assumption about a variable type.
V15.3.6	Verify that JavaScript code is written in a way that prevents prototype pollution, for example, by using Set() or Map() instead of object literals.
V15.3.7	Verify that the application has defenses against HTTP parameter pollution attacks, particularly if the application framework makes no distinction about the source of request parameters (query string, body parameters, cookies, or header fields).
V16.1.1	Verify that an inventory exists documenting the logging performed at each layer of the application's technology stack, what events are being logged, log formats, where that logging is stored, how it is used, how access to it is controlled, and for how long logs are kept.
V16.2.1	Verify that each log entry includes necessary metadata (such as when, where, who, what) that would allow for a detailed investigation of the timeline when an event happens.
V16.2.2	Verify that time sources for all logging components are synchronized, and that timestamps in security event metadata use UTC or include an explicit time zone offset. UTC is recommended to ensure consistency across distributed systems and to prevent confusion during daylight saving time transitions.
V16.2.3	Verify that the application only stores or broadcasts logs to the files and services that are documented in the log inventory.
V16.2.4	Verify that logs can be read and correlated by the log processor that is in use, preferably by using a common logging format.
V16.2.5	Verify that when logging sensitive data, the application enforces logging based on the data's protection level. For example, it may not be allowed to log certain data, such as credentials or payment details. Other data, such as session tokens, may only be logged by being hashed or masked, either in full or partially.
V16.3.1	Verify that all authentication operations are logged, including successful and unsuccessful attempts. Additional metadata, such as the type of authentication or factors used, should also be collected.
V16.3.2	Verify that failed authorization attempts are logged. For L3, this must include logging all authorization decisions, including logging when sensitive data is accessed (without logging the sensitive data itself).
V16.3.3	Verify that the application logs the security events that are defined in the documentation and also logs attempts to bypass the security controls, such as input validation, business logic, and anti-automation.
V16.3.4	Verify that the application logs unexpected errors and security control failures such as backend TLS failures.
V16.4.1	Verify that all logging components appropriately encode data to prevent log injection.
V16.4.2	Verify that logs are protected from unauthorized access and cannot be modified.
V16.4.3	Verify that logs are securely transmitted to a logically separate system for analysis, detection, alerting, and escalation. The aim is to ensure that if the application is breached, the logs are not compromised.
V16.5.1	Verify that a generic message is returned to the consumer when an unexpected or security-sensitive error occurs, ensuring no exposure of sensitive internal system data such as stack traces, queries, secret keys, and tokens.
V16.5.3	Verify that the application fails gracefully and securely, including when an exception occurs, preventing fail-open conditions such as processing a transaction despite errors resulting from validation logic.
SBD-AC-01	All communications use TLS/mTLS; service identity is verified (e.g., mesh-issued certs).
SBD-AC-02	Central IdP (OIDC/OAuth2) is used; MFA enforced on privileged/admin paths; tokens are short-lived.
SBD-AC-03	RBAC/ABAC governs APIs and publish/consume on messaging; least privilege is applied.
SBD-AC-05	Secrets are stored in a secret manager; keys/certs rotate automatically; no secrets in code/logs.
SBD-AC-06	Applicable regulatory controls (e.g., privacy/financial) are identified with evidence of design alignment.

requirement	what it asks for
SBD-AS-07	Startup resilience: services handle missing dependencies; autoscaling defined with minimum replicas.
SBD-DM-02	Encryption in transit (TLS/mTLS) and at rest with managed keys and rotation is designed in.
SBD-DM-01	Data are classified with named owners; controls are proportional to classification.
SBD-DM-05	Retention/deletion policies exist per data class; minimization is applied to collection/storage.
SBD-MT-02	Metrics/SLOs and dashboards cover service health and event-flow anomalies; alerts are actionable.
SBD-MT-03	ASVS-aligned security tests and negative tests from threat-modeling are planned in the test strategy.
SBD-MT-04	SbD controls are verifiable in test (isolation, mTLS, authZ denials, rate-limit behavior).
SBD-MT-05	OpenAPI/AsyncAPI and event catalogs are published; ADRs/runbooks are current.
SBD-MT-06	A documented Incident Response plan (roles, comms, evidence handling) exists and is rehearsed.
SBD-MT-07	Audit log retention >=12 months with tamper-evidence and a searchable hot window is defined.
SBD-MT-01	Structured centralized logs with correlation/trace IDs; administrative access is logged.
SBD-RR-06	Timeouts on all calls; health probes; explicit failover/multi-AZ strategies for critical paths.
SBD-RR-08	Caching/CDN/back-pressure patterns are applied where appropriate; performance guardrails are defined.
SBD-RR-01	Retries use exponential backoff + jitter; default exception handling and safe error models exist.
SBD-RR-05	All mutating endpoints enforce an Idempotency-Key; concurrency controls (locks/optimistic concurrency) are in place.
SBD-RR-07	Quotas and autoscaling are defined; rate-limits enforced at edges; minimum capacity documented.
C7.2.1	Verify that the system assesses the reliability of generated answers using a confidence estimation method.
C7.2.2	Verify that the application automatically blocks answers or switches to a fallback message if the confidence score drops below a defined threshold.
C12.2.2	Verify that behavioral anomaly detection identifies unusual conversation patterns, excessive retry attempts, or probing behaviors.
C12.2.3	Verify that custom rules detect AI-specific threat patterns for coordinated jailbreak attempts, prompt injection, and system prompt extraction attempts.
C12.2.4	Verify that extraction-alert events include offending query metadata to support investigation.