

SECURITY NOTES / 02

From Signed Request to Safe Processing

An evidence-led review of a webhook consumer's trust boundaries.

MODEL

Actors, assets, signed bytes, and event identity.

OBSERVE

Local tests for integrity, freshness, concurrency, and rollback.

QUALIFY

Explicit limits on provider compatibility and external effects.

Amy V. | October 2026

Independent portfolio work. Not commissioned by, affiliated with, or endorsed by Cracken. The examples describe a local reference implementation.

Research question

What must a receiver establish before an authenticated webhook changes application state?

The question deliberately begins after a request reaches the application. It avoids implying that one verifier can secure DNS, TLS, the sender's infrastructure, the shared secret, and the downstream business system. The reference implementation examines message integrity, freshness, event identity, and one local transaction. Each conclusion is tied to a defined behavior rather than a general security score.

System and adversary

A fictional sender emits `invoice.paid` events. The receiver knows one shared key and owns one SQLite database. An observer can capture and replay a request. An active adversary can alter the request but cannot recompute its HMAC without the key. Separately, a legitimate sender can retry or reuse an ID incorrectly. The latter is modeled with a correctly signed conflicting payload.

The primary asset is the receiver's state: the event record and the simulated business effect. Secondary assets include the secret and the meaning of diagnostic output. An ambiguous success label is itself a reliability risk because it can make an incomplete action look complete to an operator.

No live sender, public endpoint, or real invoice system was tested. The fixture key is deliberately public test data. A compromised key, malicious authorized sender, or incorrectly authorized business operation is not prevented by the demonstrated HMAC check.

Trust boundaries

The first boundary is between raw input and an authenticated message. The receiver restricts timestamp and event-ID formats, caps the body at 64 KiB, and compares the expected signature before JSON parsing. The second boundary is time: the signed timestamp must fall within the stated interval. The third is meaning: strict JSON and the fictional business schema reject inputs that are authentic but unusable.

The final boundary is state. The SQLite consumer inspects the event identity and digest, then commits its event record and local simulated effect together. This is where the design moves beyond a stateless cryptographic example.

Observation 1: valid authentication is repeatable

The same valid fixture passes the stateless verifier twice. This is the expected result for unchanged bytes under the same key. It reproduces a familiar failure mode in integration design; it is not a new cryptographic vulnerability.

The freshness tests use signatures calculated for their old and future timestamps. Consequently, an `outside_window` result actually tests a time decision on an authenticated message. Separate tampering tests alter signed metadata without re-signing and expect authentication failure.

The distinction matters for diagnostic writing. "Invalid webhook" conceals which boundary failed. Internally, structured reason codes make the test and its intended control inspectable. A deployed service would still need to decide which diagnostics to expose to an untrusted caller and which to retain

in protected logs.

Observation 2: event identity needs a content contract

An identical event ID and body digest produces `duplicate_completed` after an earlier local completion. The same ID with changed, correctly signed content produces `event_conflict`. The original database record is preserved.

The implementation intentionally treats byte changes as content changes. That includes JSON whitespace changes under the same ID. Whether that is appropriate for a provider depends on its stable-payload and retry contract; the lab does not assume all vendors preserve identical serialization. A real adapter must define identity and conflict semantics before adopting this policy.

One database is scoped to one sender. Another sender cannot safely share the same event-ID namespace without an authenticated namespace extension. Likewise, two different event IDs for one business object remain two events. Domain-level uniqueness is an additional rule, not a property of event-ID deduplication.

Observation 3: completion must be earned

The event and effect rows share an SQLite transaction. Failure injection before the effect and after its insertion verifies that an exception before commit leaves neither row committed. A subsequent attempt can then complete. Sixteen simultaneous deliveries produce one local effect in the tested execution.

The important qualification is “local.” The effect is an SQLite row, so it participates in the transaction. Sending an email or creating a payment would cross another system boundary. This implementation does not make that remote operation atomic with its local event record.

```
Authentication -> Freshness -> Schema -> Transaction
                                     | event row
                                     | local effect row
                                     + commit together

Remote API call: outside the demonstrated transaction
```

Reopening a consumer object is tested. Operating-system crashes, power interruption, distributed workers, and storage faults are not established by those tests. SQLite's own durability behavior has filesystem and hardware assumptions; the local experiment should not be presented as independently validating them.

Evidence map

- Exact bytes: whitespace and key-order tampering tests.
- Verification order: parser spy test for an invalid signature.
- Freshness: validly signed stale/future fixtures and inclusive boundary tests.
- Parsing: malformed UTF-8, duplicate keys, non-object JSON, and non-finite constants.

- Identity: identical retry, conflicting reuse, and newly signed retry.
- State: reopened database, simultaneous connections, and rollback around the local effect.

Run `python -m unittest -v` and `python demo.py`. The evidence directory records the tested source hashes, environment, raw test output, and scenario outcomes. This map describes the implemented checks; it does not imply exhaustive security coverage.

Questions before a provider integration

Ask what bytes the provider signs, whether timestamps are regenerated on retry, and how event IDs are scoped. Establish whether identical logical events can have different serialization. Determine the maximum retry horizon, skew allowance, and consequences of deleting old records. Identify the business effect that must be idempotent and what evidence exists after an ambiguous response.

Then test those answers against the provider's current contract. The custom newline-delimited envelope in this lab is intentionally explicit; it is not a drop-in adapter or RFC 9421 implementation.

Conclusion

The useful finding is a separation of responsibilities. A signature establishes message integrity under a shared-key assumption. A timestamp constrains age. An event record supplies history. A transaction defines which local changes succeed together. None substitutes for authorization or establishes exactly-once remote behavior.

This separation is also a writing method: state the boundary, reproduce the behavior, identify the remaining risk, and tell the reader what decision to make next.

References

- [OWASP Webhook Security Cheat Sheet](#)
- [Python hmac documentation](#)
- [SQLite transactions](#)
- [SQLite atomic commit and assumptions](#)
- [RFC 9421: HTTP Message Signatures](#)