SMARTFLOW OBSERVATORY

Fix Your 402

Free sample: what buyer-agents check before paying

Tom Smart
smartflowproai.com

Edition 2026.08

14. Chapter 9: What buyer-agents check before paying

A buyer-agent is not a human with a wallet extension and patience. It is software with a task, a spending policy, a deadline, and alternative providers.

There is no single universal buyer implementation, but a careful buyer usually follows the same gates.

The machine buying loop

1. **Discover:** Find a resource that appears able to complete the task.
2. **Match:** Compare method, input, output, network, and displayed price with the task policy.
3. **Probe:** Call the resource without payment and expect a `402` challenge.
4. **Parse:** Decode the canonical payment requirements.
5. **Validate:** Check version, scheme, network, asset, amount, recipient, timeout, and resource binding.
6. **Compare:** Confirm the live quote is compatible with discovery and the buyer's budget.
7. **Authorize:** Sign the payment payload for the accepted requirements.
8. **Retry:** Send the same resource request with the payment signature.
9. **Settle and receive:** Verify the payment response and consume the resource.
10. **Record:** Save the quote, result, and receipt for retries, disputes, and provider scoring.

Every chapter in this guide protects one of those gates.

Gate 1: Can the discovery system understand the service?

For CDP Bazaar, the buyer and facilitator need a live `402` whose `extensions.bazaar.info.input` describes how to call the route and whose `schema` validates that advertised information. The `resource` object supplies the resource URL and may include service metadata. Other crawlers may also inspect the optional `/.well-known/x402` convention.

Failure patterns:

- Missing or malformed `extensions.bazaar`
- Required input represented by empty or invalid placeholders
- Advertised method or body shape rejected before payment middleware
- Stale or dead resource URL
- Optional manifest returns HTML, bare strings, or old terms

What improves selection:

- A structurally valid Bazaar extension in the live challenge
- Concrete input values the route accepts
- A truthful output example
- Stable URLs
- A route description that says what the buyer receives
- An optional compact public manifest for crawlers that use the convention

Gate 2: Can I safely parse the live quote?

For v2, the buyer expects a Base64 JSON `PaymentRequired` object in `PAYMENT-REQUIRED`. It does not need to guess a custom header name or scrape a prose error message.

Failure patterns:

- No `402`
- Undecodable header
- Empty `accepts`
- Body-only response presented as v2
- Mixed version fields
- Unsupported scheme

What improves selection:

- Canonical v2 headers
- One clear protocol version
- Non-empty payment requirements
- A resource URL that matches the request
- Standard SDK output instead of a hand-built near-match

Gate 3: Do the economics fit policy?

The buyer checks the atomic amount, asset, network, and scheme. It may compare the live quote with discovery, a cached quote, and a maximum spend rule.

Failure patterns:

- Display price copied into `amount`
- Decimal or currency symbol in a digit-string field
- v1 amount key in a v2 challenge
- Wrong asset for the selected network
- Live quote above the buyer's limit

What improves selection:

- Exact integer conversion
- Clear asset and CAIP-2 network
- Stable or explicitly dynamic pricing
- No disagreement between header and compatibility body

Gate 4: Am I paying the expected recipient?

The buyer compares `payTo` across discovery, the live challenge, previous successful calls, and any allowlist it maintains.

Failure patterns:

- Manifest and challenge disagree
- Different instances return different recipients
- Recipient changes without a terms version or effective date
- A default wallet appears after a tenant lookup fails

What improves selection:

- One internal source of truth
- Versioned recipient rotation
- Consistent checksummed or normalized representation
- Archived post-deploy validation

Gate 5: Will the offer remain the same long enough to pay?

The buyer may re-probe before authorizing or compare the accepted requirements with the current challenge. It wants to know that it will not sign one offer and have the server apply another.

Failure patterns:

- Terms change between adjacent probes on a fixed-price route
- Cache returns old discovery after a live change
- Multiple instances serve different configuration
- A scanner result cannot be reproduced from raw evidence

What improves selection:

- Versioned terms
- Quote binding
- Explicit validity windows
- Raw response archives
- Clear cache policy

Gate 6: Can the endpoint complete the handshake in time?

The buyer treats timeouts and server errors as routing signals. It may try another provider instead of debugging yours.

Failure patterns:

- Cold-start timeout
- `5xx` during a burst of probes
- Input validation returns `400` before payment middleware
- A protected upstream runs before payment
- Published method never reaches `402`

What improves selection:

- A fast challenge path
- Correct method and valid input examples
- No settlement work before a payment payload exists
- Consistent behavior across repeated probes

Gate 7: What happens after I pay?

This guide focuses on pre-pay conformance, but a buyer also cares about post-pay behavior:

- Does the retry bind to the same resource and input?
- Is the payment accepted only once where replay protection requires that?
- Does the server return the promised output type?
- Is a settlement response exposed through the standard channel?
- Are paid errors handled according to the published policy?

A green pre-pay scan does not answer all of these. Add a controlled end-to-end payment test before launch, and make its spend limit explicit.

Why “works for me” fails a machine customer

A human can refresh, inspect DevTools, ask in Discord, or decide that two nearly identical fields probably mean the same thing. A buyer-agent should treat ambiguity as financial risk.

Machines are especially strict about:

- Location: header versus body
- Type: integer versus string
- Unit: displayed currency versus atomic amount
- Identity: one recipient versus another
- Time: current response versus stale snapshot
- Method: the route that was documented versus the route that returned `402`

This strictness is useful. It tells you exactly what to make deterministic.

The pre-pay checklist

Before asking an agent to buy, answer yes to each question:

- Does the live `402` include `extensions.bazaar.info.input` with the correct method and concrete accepted values?
- Does the Bazaar `schema` validate the advertised `info` object?
- If you serve the optional `/.well-known/x402` convention, can a crawler fetch it without credentials?
- If you serve that manifest, does every listed resource have a URL, method, description, and terms?
- Does the documented input reach the payment middleware?
- Does an unauthenticated request return HTTP `402`?
- Does v2 use a decodable `PAYMENT-REQUIRED` header?
- Is `x402Version` the integer `2`?
- Is every v2 network a CAIP-2 identifier?
- Is `amount` a digit string in atomic units?
- Are `asset`, `payTo`, and `maxTimeoutSeconds` present?
- Does the challenge bind to the requested resource?
- Do discovery and challenge agree on network, economics, and recipient?
- Do repeated probes of fixed terms agree?
- Does the challenge path stay inside your latency budget without `5xx`?
- Can you reproduce the verdict from an archived raw response?

What a seller should publish

Buyer-agents should not need a private conversation to learn the basics. Publish:

- A live `402` with `extensions.bazaar` for CDP Bazaar discovery
- `/.well-known/x402` only if you support the optional community convention
- OpenAPI or another machine-readable input and output contract
- Exact resource methods
- Supported networks and assets
- A valid request example for each route with required input
- A short pricing-change policy
- A support or issue URL
- A changelog for breaking payment-term changes

Keep marketing prose separate from the machine contract. A description can help selection,

but it cannot replace required fields.

Run the buyer's test before the buyer does

The validator is a pre-flight check, not a badge generator. Run it on the deployed URL after every change to route, middleware, manifest, pricing, asset, recipient, proxy, or runtime.

A useful trust signal is not one green screenshot. It is the ability to reproduce the same coherent contract from fresh public probes and explain every intentional change.

Inside the full guide

The paid edition is a 48-page implementation guide organized into six parts and 21 sections:

- **Part I, Start here (1-5):** the repair loop, the two JSON shapes, validator setup, scan reports, and the evidence index.
- **Part II, Discovery and payment challenge errors (6-10):** header transport, CAIP-2, `extensions.bazaar`, the optional `/.well-known/x402` convention, amount keys, and mixed v1/v2 signals.
- **Part III, Identity and drift errors (11-12):** `payTo` drift and differences between scan results and the live paywall.
- **Part IV, Behavior under probe (13):** latency, timeouts, and `5xx` responses during the challenge path.
- **Part V, Getting chosen (14):** the complete buyer-agent pre-pay chapter included in this sample.
- **Part VI, Release reference (15-21):** the one-page validator checklist, what a green scan proves, when to stop doing it yourself, FAQ, references, licence, support, and changelog.

Each fault-focused chapter follows the same loop: see the symptom, confirm it with one command, apply a before-and-after fix, and verify the result.

Get the complete guide for $29:
https://verify.smartflowproai.com/guide/