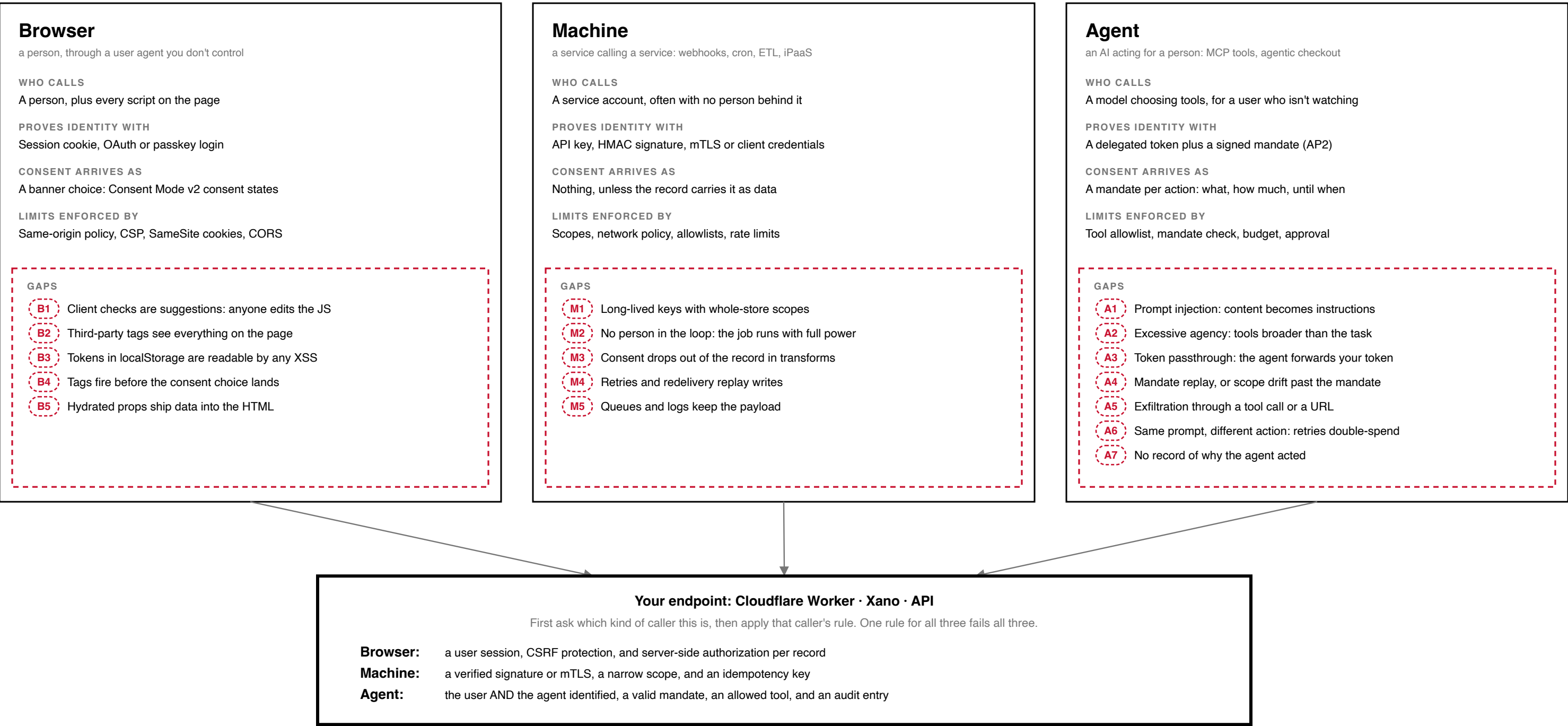


BROWSER vs MACHINE vs AGENT — THREE CALLERS, THREE WAYS TO FAIL

who is calling, what proves it, what limits it, and where each one breaks

Red dashed = a gap. B = browser, M = machine, A = agent. Numbers match the list below.

THREE CALLERS, ONE ENDPOINT — THE SAME API CAN'T TRUST THEM THE SAME WAY



Your endpoint: Cloudflare Worker · Xano · API

First ask which kind of caller this is, then apply that caller's rule. One rule for all three fails all three.

Browser:

a user session, CSRF protection, and server-side authorization per record

Machine:

a verified signature or mTLS, a narrow scope, and an idempotency key

Agent:

the user AND the agent identified, a valid mandate, an allowed tool, and an audit entry

WHERE EACH ONE FAILS

CONCERN	BROWSER	MACHINE	AGENT
Fails open when	A check lives only in client JavaScript	A key outlives its job or its scope	Tool output is read as an instruction
Replay looks like	A resubmitted form or a stolen session	A redelivered webhook or a re-run batch	A retried tool call or a reused mandate
Data leaks through	Third-party tags, page source, caches	Queues, logs, iPaaS error records	Prompts, tool calls, model context
Consent is checked	Before any tag loads	On every record, as a field	Before every side effect, as a mandate
Audit must answer	Which user, which session?	Which key, which job, which payload?	Which user, agent, mandate and tool, and why?

ASSUMPTIONS THAT BREAK AS YOU GROW — FROM A JVM THAT NEVER LEAVES THE VM TO A SYSTEM AI CAN READ

ASSUMPTION	A JVM THAT NEVER LEAVES THE VM	WHEN YOU ADD COMMERCE	WHEN YOU GO GLOBAL	WHEN YOU ADD LLMS: AI-LEGIBLE
Trust boundary	The VM is the wall; inside is trusted	Payments, tax and tag scripts run outside it	Clouds and regions multiply the walls	Your content is input to someone else's model
Identity	One app, one user table, one session store	Customers, guests, staff, payment tokens	Local logins and age rules per market	An agent acts for a user: identify both
Where data lives	In the heap and one database	Card data enters PCI scope; webhooks leave	Residency: where each copy lives is legal	Prompts, embeddings and model logs are copies
Consent	Not modeled: the app owns its data	Marketing consent and order data diverge	GDPR, CPRA, PIPA, CASL: defaults differ	Training and retrieval need their own yes
Time and money	One clock, one currency, one locale	Retries and refunds need idempotency keys	Currencies, tax, FX, time zones, languages	An agent can spend: mandate, amount, expiry
Failure	An exception stays in the log	A failure can charge twice or lose an order	One region down mustn't take the others	A plausible wrong answer looks like success
Audit asks	Who logged in?	Who paid, refunded or changed a price?	Which law applied, and where did data go?	Which user, agent, mandate and source?
Readable by	Your own team	Auditors, the PCI assessor, payment partners	Regulators in every market	Crawlers, answer engines and agents

Each column keeps every problem to its left. The AI column is red dashed: making a system legible to machines also makes it readable, quotable and steerable by them.

SEVENTEEN GAPS — AND THE CONTROL THAT CLOSES EACH

<div><div>B1</div><div>Client checks are suggestions</div><div>Fix: Authorize every request on the server, per record</div></div>	<div><div>M1</div><div>Keys too broad, too long</div><div>Fix: Least-privilege scopes, short lifetimes, rotation</div></div>	<div><div>A1</div><div>Prompt injection</div><div>Fix: Treat fetched content as data; approve side effects outside the model</div></div>
<div><div>B2</div><div>Third-party tags see the page</div><div>Fix: CSP, Subresource Integrity, a script inventory (PCI DSS 6.4.3)</div></div>	<div><div>M2</div><div>No person in the loop</div><div>Fix: Act as a named service, log it, cap what it can change</div></div>	<div><div>A2</div><div>Excessive agency</div><div>Fix: One narrow tool per task; read and write as separate tools</div></div>
<div><div>B3</div><div>Tokens readable by XSS</div><div>Fix: HttpOnly, Secure, SameSite cookies; nothing in localStorage</div></div>	<div><div>M3</div><div>Consent drops out</div><div>Fix: Carry consent state as a field; refuse records without it</div></div>	<div><div>A3</div><div>Token passthrough</div><div>Fix: Audience-bound tokens per tool; never forward the user's token</div></div>
<div><div>B4</div><div>Tags fire before consent</div><div>Fix: Consent Mode default denied; load tags after the choice</div></div>	<div><div>M4</div><div>Replays duplicate writes</div><div>Fix: Verify the HMAC on the raw body; one idempotency key per event</div></div>	<div><div>A4</div><div>Mandate replay or drift</div><div>Fix: Sign amount, merchant, expiry and nonce; check every call</div></div>
<div><div>B5</div><div>Hydration ships data</div><div>Fix: Pass only what the screen shows; fetch PII after consent</div></div>	<div><div>M5</div><div>Payloads persist</div><div>Fix: Store IDs, not payloads; purge queues; scrub logs</div></div>	<div><div>A5</div><div>Exfiltration by tool</div><div>Fix: Egress allowlist; no raw-URL tools; filter outputs for PII</div></div>
		<div><div>A6</div><div>Retries double-spend</div><div>Fix: An idempotency key on every side-effecting tool call</div></div>
		<div><div>A7</div><div>No record of why</div><div>Fix: Log user, agent, mandate, tool and inputs to an append-only ledger</div></div>

A browser is a person whose code you can't trust. A machine is code with no person. An agent is code choosing what to do for a person. Each needs its own rule at the same endpoint.