OPNLI — OPEN CONSENT PROJECT

The Pied Piper Roadmap

openlytrusted.ai/pied-piper

Version 1.0 — September 23, 2026

The one-paragraph version

THE SHORT VERSION

Google, Meta, Microsoft, and Apple are not building the agent economy from scratch. They are extending the platforms they already own — browsers, operating systems, app stores, social networks — into a world where AI agents act on behalf of humans. The two open protocols that make this possible, MCP and WebMCP, are well-designed standards for how agents discover and call tools. What they deliberately leave out is how humans say yes or no. The Human Consent Layer fills that gap — and the developers who adopt it now, before the architecture hardens, define what "trustworthy" means for everyone who comes after.

This is the Pied Piper pattern: the music is enchanting, the children follow willingly, and by the time anyone notices where they're going, the architecture is set.

Part 1: The Mobile Playbook — and Why It's Happening Again

In 2008, Apple launched the App Store.

Apple did not build every application. Apple built the platform every application ran on — the operating system, the browser, the distribution channel, the payment system, the review process. Developers built the apps. Users used the apps. Apple controlled the experience between the developer and the user.

The value exchange was real. Developers got distribution to hundreds of millions of devices. Users got a curated, secure app ecosystem. Apple got a 30% commission and structural control over what software could run on the most popular computing device in history.

Google replicated this with Android and the Play Store. Facebook replicated it with the social graph and the platform API. Amazon replicated it with the marketplace. The playbook was always the same: build the platform, open it to developers, let developers create the value, control the user's experience of that value.

The agent economy is the next iteration of this playbook. The technology has changed — AI agents instead of mobile apps, tool-calling protocols instead of app APIs, browser-based assistants instead of app stores. The structure has not.

The companies that own browsers, operating systems, and cloud platforms are building AI agents directly into those platforms. The open protocols that let agents discover and call tools mean that every developer's tool becomes a capability of the platform's agent. The user interacts with the platform's agent, not with the developer's tool. The platform controls the experience — including when, how, and whether to ask the human for consent.

This is the Pied Piper pattern: the music is enchanting, the children follow willingly, and by the time anyone notices where they're going, the architecture is set.

The Human Consent Layer (HCL) ensures humans always have a choice.

Part 2: What MCP and WebMCP Actually Do — in Plain Language

Two protocols are standardizing how AI agents find and use tools.

Both are well-designed. Both are open. Both solve real problems. And both deliberately leave one problem for someone else to solve.

MCP — the Model Context Protocol

MCP is for AI agents that run on a server or on your local machine — a desktop agent like CROCbox, a terminal tool like Claude Code, or a cloud-hosted enterprise agent. The agent connects to an MCP server (a small program that exposes specific capabilities as "tools"), discovers what tools are available, and calls them. This happens machine-to-machine. No browser is involved.

An MCP server might expose tools like "search this person's email," "query this database," "create a calendar event," or "read health records from this system." The agent discovers these, decides which to call, calls them, gets results, and uses them to accomplish a task.

MCP is governed by the Linux Foundation's Agentic AI Foundation. Its GitHub repository has over 40,000 stars. Anthropic originated the protocol and contributed it to open governance. Every major AI lab supports or is building support for MCP.

The MCP specification states: "For trust & safety and security, there should always be a human in the loop with the ability to deny tool invocations." The keyword is **SHOULD** — an RFC 2119 advisory, not a mandate.

> **THE GAP**
>
> The protocol does not define how, when, or whether the agent stops to ask. There are no standing rules, no audit chain, no revocation mechanism at the protocol level.

WebMCP — the Web Model Context Protocol

WebMCP is for AI agents that run inside the browser. When you visit a website, that website can register tools using `document.modelContext.registerTool()` — capabilities like "book a table," "order prints," or "view medical records." A browser-based AI assistant discovers those tools and can call them on your behalf.

WebMCP is a W3C Community Group draft specification, led by engineers from Microsoft and Google. Chrome shipped the WebMCP Origin Trial in Chrome 149, making the API available to developers for testing.

WebMCP added a `consequentialHint` property — a boolean flag (default false) that tells the browser agent "this action is important, maybe ask the user before proceeding." The hint is advisory. The agent can ignore it.

What both protocols share

Both protocols solve how an agent discovers tools, how it calls them, what the input and output formats look like, and how errors are handled.

Here is what both protocols deliberately leave unsolved: how a human approves or denies an action, when the agent must stop and ask, what record exists of what was approved, how a human revokes a prior approval, whether the human can set standing rules that persist across sessions, and whether there is a tamper-evident log of decisions.

Both protocol communities know this. The WebMCP community debated consent for months in their GitHub issues. The MCP specification acknowledged the gap with a SHOULD. A developer proved the gap empirically (see Part 4). The consent problem is not a secret — it is a deliberate scope boundary.

> Both protocols chose to leave consent enforcement to “someone else.”

The question is: who is “someone else”?

Today, the answer is the platform that owns the browser or the agent runtime. That is the Pied Piper pattern.

The Human Consent Layer offers an alternative path so humans always have a choice.

Part 3: The Pied Piper Roadmap — Platform by Platform

Here is how each major platform uses these open protocols to extend the silos they already own.

Question	Google	Meta	Microsoft	Apple
Who decides when to ask	Google / Chrome	Meta / Sentinel	Microsoft / Copilot	Not stated
Who writes the rules	Not stated	Meta sets the menu	Not stated	Not stated
Who holds the record	Not stated	No user-held tamper-evident record	Not stated	Not stated
Training on user data	Mandatory for Spark; no opt-out	On by default; opt-out	Not stated	Not stated

THE PATTERN

The platform owns the runtime (browser, OS, app ecosystem). The open protocol (MCP or WebMCP) lets developers register tools. The platform’s AI agent discovers and calls those tools. The platform decides when and how to ask the user for consent. The developer who built the tool has no say in the consent experience. The user has no independent, verifiable record.

The music is enchanting. The tools are useful. The agents are helpful. And the human’s structural position — their ability to see, decide, and verify — is being architected away.

Google: Chrome + Gemini + Spark

Google owns Chrome, used by approximately 65% of web users worldwide. Chrome is shipping the WebMCP API through Origin Trials. When a website registers tools using WebMCP, Chrome's built-in AI assistant (powered by Gemini) can discover and call those tools.

The developer built the tool. The user sees Chrome's assistant using it. Google controls the experience between the tool and the user — including when, or whether, to ask for consent.

In September 2026, Google launched Spark — an always-on agent that runs 24/7 on Google Cloud, accessing the user's email, calendar, files, and browser. Google's television campaign "Instant Replay" markets comfort: agents that handle the tedious stuff so you don't have to think about it.

Google's Spark onboarding disclaimer states it "may share your info or make purchases without asking." Model training on user data is mandatory, with no opt-out. Google will not ship Spark in the EU — the one jurisdiction whose regulations would require showing users what the agent is doing before it does it.

Meta: Muse + WhatsApp + Instagram

Meta launched Muse — a personal agent with its own secure virtual machine, browser, and payment integration, distributed through WhatsApp and Instagram, reaching #1 on the App Store in its first week.

Meta built Sentinel, the best agent interception architecture in the industry. Sentinel is a separate host-side agent that serves as the sole permission authority for every action Muse takes. It intercepts every connector call, evaluates network requests at layers 4 and 7, uses eBPF-based taint tracking to flag processes that have read user data, and stores credentials in a separate authentication daemon the agent never sees.

This is serious, impressive engineering. But Sentinel decides the menu. The user picks from what Meta offers. The user does not write their own rules, does not hold a tamper-evident record of what was approved, and cannot independently verify decisions. Training is on by default with an opt-out toggle — not a structural consent boundary.

> "The isolation gets engineered carefully and the consent surface gets designed last."

Microsoft: Copilot + Windows + Edge + Azure

Microsoft owns Windows (over 70% of desktop operating systems), Edge (growing browser share), and Azure (major cloud platform). Microsoft's Copilot is being integrated into Windows as a system-level agent with access to local files, applications, and services.

MCP servers running on Azure or locally become capabilities of Copilot. Developers build MCP servers. Copilot discovers and calls their tools. The user interacts with Copilot, not with the developer's server directly. Microsoft controls the consent surface.

Apple: Siri + Safari + the OS

Apple controls Safari, iOS, and macOS. When WebMCP reaches Safari (or Apple ships its own equivalent), websites that register tools become capabilities of Siri or whatever agent Apple deploys. Apple has the most restrictive platform policies in the industry — and historically the most opaque about how data flows through its agent features.

Part 4: Where the Human Disappears

The human disappears in the gap between tool discovery and tool execution.

The WebMCP gap

A website registers a tool. The browser's AI discovers it. The browser's AI decides to call it. What happens between "decides to call" and "calls"?

WebMCP says: nothing mandatory. The consequentialHint is advisory. The agent can proceed without asking. The website developer has no mechanism to require consent. The user has no mechanism to see what happened.

A developer in the WebMCP community proved this gap empirically. In Issue #165 on the WebMCP GitHub repository, the developer ran a controlled experiment: same tool, same model, same system prompt. In one arm, the tool's description asked the model to wait for human approval before executing. In the other arm, a structural code-level gate enforced the wait. Across eighteen trials, the structurally enforced gate produced zero unauthorized executions out of eighteen. The description-only approach produced eighteen unauthorized executions out of eighteen.

In one trial a model narrated its own reasoning for bypassing the approval request — while bypassing it.

> "Whatever this group specifies, if the guarantee lives in prose that the model reads, it is not a guarantee. That argues for the resolution being something the user agent or the page can enforce structurally."

This is not a theoretical concern. It is a measured result. The consent gap is not a bug — it is an architectural absence.

The MCP gap

An agent connects to an MCP server. The server exposes tools. The agent decides to call a tool. MCP says the agent "SHOULD" get human approval. The agent's client runtime decides whether to enforce this.

Most client runtimes do not enforce it consistently, because enforcement would require defining when to ask (every call? only "consequential" ones? who decides what's consequential?), defining what options to present (allow once? allow always? deny? time-limited?), recording the decision in a way the

human can verify, and honoring revocation immediately. None of this is specified. Each client makes its own choices — or makes no choice, which is also a choice.

What disappearing looks like

The human's position in both flows is identical: they are downstream of a decision the platform already made.

The platform decided when to ask.

The platform decided what options to present.

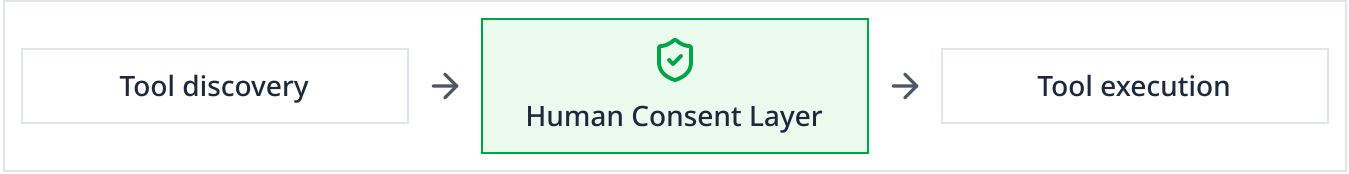
The platform holds the record.

> **THE RECORD**
>
> The human's "yes" is stored in the platform's database — visible to the platform, invisible to the human, unverifiable by anyone outside.

Part 5: Where the Green Shield Intercepts

The Human Consent Layer intercepts at the exact point where the human disappears — between tool discovery and tool execution.

The machine must declare what it's about to do, and ask, before it acts. A human can't consent to what they can't see.

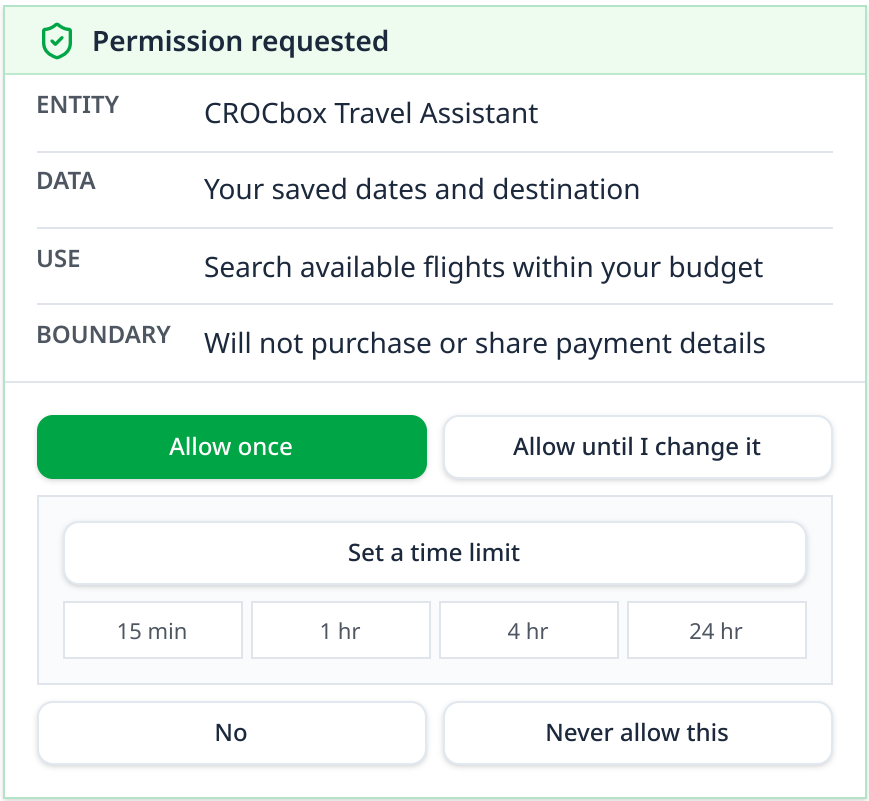


In the WebMCP flow

A developer adds three script tags to their website and wraps their `registerTool()` call with `WebMCPHCL.registerTool()`:

```HTML
<script src="https://mikeoz.github.io/webmcp-hcl/src/audit-log.js"></script>
<script src="https://mikeoz.github.io/webmcp-hcl/src/consent-card.js"></script>
<script src="https://mikeoz.github.io/webmcp-hcl/src/webmcp-hcl.js"></script>
```

The tool is still registered with the browser's WebMCP API — any browser agent can still discover it. But when the agent tries to call the tool, the bridge intercepts. The Green Shield consent card fires. The user sees four pieces of context — Entity (who is asking), Data (what data is involved), Use (what the tool does with it), Boundary (what it will not do) — and makes one of five choices: Allow once; Allow until I change it (standing allow); Set a time limit (15 minutes, 1 hour, 4 hours, 24 hours); No (deny once); or Never allow this (standing deny).



The agent cannot bypass the gate. The decision is recorded in a SHA-256 hash-chained audit log in the browser. Standing rules persist across page refresh. The user can verify chain integrity and revoke any rule at any time.

The website owner — not Google, not the browser — controls the consent experience. The developer who built the tool decided that consent is required. The platform's agent must comply.

Zero dependencies. Pure browser JavaScript. 863 lines. Apache 2.0 licensed.

In the MCP flow

A developer adds the HCL consent gate to their MCP server or agent runtime using the HCL DevKit. The agent presents CARDS (Entity, Data, Use, Boundary) to the OTN Consent Server. The Consent Server fires the Green Shield on whatever surface the human is using — a chat interface, a desktop app, a mobile client. The human makes one of the same five choices. The decision is recorded in the hash chain. A cryptographic consent proof (EdDSA JWT) is minted. The agent can only proceed with that proof.

CROCbox is the reference implementation for MCP-based consent.

The structural properties — in both flows

These four properties are what distinguish consent from permission:

Consent Before Execution. The agent tells the human what it will do before it does it. Not "designed to ask" — required to ask. Enforced by code, not requested by text. The ablation study's 0/18 result is the design specification.

Binding, Standing Rules. The human sets boundaries once. Inside those boundaries, the agent acts without asking — because the human already answered. Outside those boundaries, the agent stops. This solves consent fatigue honestly: not by removing consent, but by making one act of consent cover many future actions, revocable at any time.

Tamper-Evident Record. Every decision is recorded in a SHA-256 hash chain where each entry links to the previous one. The human can verify the chain. The platform cannot alter it after the fact.

Verifiable, Revocable. The human can see every decision, every rule, every action taken under those rules. Revocation is immediate — revoke a rule, and the agent must ask again, starting with the next action.

Part 6: What a Developer Does About It

Add three script tags.

If you build websites with tools for AI agents (WebMCP)

Replace `document.modelContext.registerTool()` with `WebMCPHCL.registerTool()`. Provide the CARD context. That is the entire integration.

```HTML
<script src="https://mikeoz.github.io/webmcp-hcl/src/audit-log.js"></script>
<script src="https://mikeoz.github.io/webmcp-hcl/src/consent-card.js"></script>
<script src="https://mikeoz.github.io/webmcp-hcl/src/webmcp-hcl.js"></script>
```

Your tool still works with Chrome's AI, with any browser agent, with any WebMCP-compatible platform. The difference: the Green Shield fires before execution. Your user sees the consent card. Your user decides. The decision is recorded.

Try the live demo or explore the WebMCP Bridge.

If you build AI agents or MCP servers (MCP)

Add the HCL consent gate to your server or runtime. The DevKit provides the audit chain library, the consent gate pattern, and schema validators.

Read the HCL Standard, use the DevKit, and study CROCbox, the reference implementation.

> **Earn the Green Shield badge**
>
> The Green Shield badge means one thing: "This tool asks before it acts, and proves it." Not "safe" or "good" — just "asks before it acts, and proves it."
>
> The certification model is Apple-style notarization, not App Store review. The developer runs the conformance checks. The system confirms the result. The badge is earned, not claimed.
>
> Follow the certification path and join the Openly Trusted Network.

The $99 Developer Starter Kit includes OTN membership, entity registration, a cloud Data Bucket, OpenRouter access with ZDR enforcement, the HCL bridge files, and the conformance checklist. $20/month ongoing.

Part 7: Why First Movers Win

In 2014, most of the web ran on HTTP — unencrypted, unverified.

The Let's Encrypt parallel

HTTPS existed but required buying a certificate, configuring a server, and renewing annually. Let's Encrypt made SSL certificates free, automatic, and ubiquitous. Within three years, HTTPS became the default. Browsers started marking HTTP sites as "Not Secure."

Let's Encrypt did not compete with Verisign by building a better certificate authority. It made the right thing the easy thing — and made the wrong thing visible.

> Let's Encrypt made the right thing the easy thing — and made the wrong thing visible.

The HCL is the same play for consent. Make structural human consent the default for every AI agent interaction. Make the integration easy — three script tags, one function call. Make the absence visible — tools without the Green Shield are tools where the agent acts without asking.

The window is now

The agent economy's consent architecture is being decided in this window — before it hardens. Chrome shipped the WebMCP Origin Trial in Chrome 149. Agents are starting to discover and call tools on live websites. MCP adoption is accelerating across every major AI platform.

Earn the Green Shield badge

The developers who add the Green Shield today are ready for the agent economy when it arrives at scale. When Chrome's AI starts calling tools on websites, the Green Shield tools will fire the consent card. The tools without it will not. The users will see the difference.

The first-mover advantage is structural: the developer who adds consent today defines what "trustworthy" means before the platforms define it for them. The platform that adopts the HCL first gets to challenge its competitors — just as Meta challenged YouTube, TikTok, and Snap to match their parental controls on children's screen time.

The Pied Piper test

Ask every agent platform three questions:

The Pied Piper Test

- ◯ Can the agent act without asking — and does the fine print say so?
- ◯ Does the user write their own rules, or only pick from a platform-curated menu?
- ◯ Does the user hold a tamper-evident record, or only the platform's rendering of its own behavior?

If the answer to any of these reveals a gap, the music is playing. The question is whether you follow it — or build something better.